Advances in Software Engineering and its Applications

Research in Computing Science

Series Editorial Board

Research in Computing Science es una publicación trimestral, de circulación internacional, editada por el Centro de Investigación en Computación del IPN, para dar a conocer los avances de investigación científica y desarrollo tecnológico de la comunidad científica internacional. **Volumen 126,** noviembre 2016. Tiraje: 500 ejemplares. *Certificado de Reserva de Derechos al Uso Exclusivo del Título* No. : 04-2005-121611550100-102, expedido por el Instituto Nacional de Derecho de Autor. *Certificado de Licitud de Título* No. 12897, *Certificado de licitud de Contenido* No. 10470, expedidos por la Comisión Calificadora de Publicaciones y Revistas Ilustradas. El contenido de los artículos es responsabilidad exclusiva de sus respectivos autores. Impreso en la Ciudad de México, en los Talleres Gráficos del IPN – Dirección de Publicaciones, Tres Guerras 27, Centro Histórico, México, D.F. Distribuida por el Centro de Investigación en Computación, Av. Juan de Dios Bátiz S/N, Esq. Av. Miguel Othón de Mendizábal, Col. Nueva Industrial Vallejo, C.P. 07738, México, D.F. Tel. 57 29 60 00, ext. 56571.

Editor responsable: *Grigori Sidorov, RFC SIGR651028L69*

Research in Computing Science is published by the Center for Computing Research of IPN. **Volume 126**, November 2016. Printing 500. The authors are responsible for the contents of their articles. Printed in Mexico City, in the IPN Graphic Workshop – Publication Office.

Advances in Software Engineering and its Applications

Matías Alvarado Mentado
Perfecto Malaquias Quintero Flores (eds.)

Instituto Politécnico Nacional, Centro de Investigación en Computación
México 2016

ISSN: 1870-4069

Instituto Politécnico Nacional (IPN)
Centro de Investigación en Computación (CIC)
Av. Juan de Dios Bátiz s/n esq. M. Othón de Mendizábal
Unidad Profesional "Adolfo López Mateos", Zacatenco
07738, México D.F., México

http://www.rcs.cic.ipn.mx
http://www.ipn.mx
http://www.cic.ipn.mx

Indexed in LATINDEX and Periodica / Indexada en LATINDEX y Periódica

Printing: 500 / Tiraje: 500

Printed in Mexico / Impreso en México

ISSN 1870-4069

Editorial

This volume of the journal "Research in Computing Science" contains selected papers on Software Engineering and its Applications in the context of the information technology industry. Software engineering can be studied and developed from different subfields such as: requirements engineering, software design, software construction, software testing, software quality, software engineering management, software development process, mathematical foundations, agile methods such as Scrum, XP and DSDM, etc. The combination and cooperation between these fields has been the key of advance in Software Engineering. This volume contains thirteen papers related to various aspects of the development and applications of different methodologies of software engineering, organized in the following sections:

- Requirements engineering,
- Software construction,
- Software testing,
- Software quality,
- Agile methods,
- Mathematical foundations, and
- Applications.

The papers in this volume have been carefully chosen by the Editorial Board based on evaluation by at least two members of the reviewing committee. The main criteria for the selection were originality and technical quality of the papers. Submission, reviewing, and selection process was supported free of charge by the EasyChair system (www.EasyChair.org).

This volume is the result of hard work and collaboration of many people. First of all, we thank the authors of the papers included in this volume for their technical excellence, which made possible the high quality of this volume. We also thank the members of the Editorial Board of the volume and the reviewing committee for their hard work on selection of the best papers out of the forty submissions we received.

Matías Alvarado Mentado
CINVESTAV–IPN, Mexico

Perfecto Malaquias Quintero Flores
Tecnológico Nacional de México/IT Apizaco, Mexico

Guest Editors

ISSN 1870-4069

Table of Contents

Page

ISSN 1870-4069

Intercesión en invocaciones de métodos con la reflexión de Java

Andoni Rodríguez-Díaz, Ulises Juárez-Martínez, Silvestre Gustavo Peláez-Camarena, Hilarión Muñoz-Contreras, Beatriz A. Olivares-Zepahua

Instituto Tecnológico de Orizaba,
División de Estudios de Posgrado e Investigación, Orizaba,
Veracruz, México

{arodriguezd,ujuarez}@ito-depi.edu.mx,sgpelaez@yahoo.com.mx,
hmunoz@itorizaba.edu.mx,bolivares@ito-depi.edu.mx

Resumen. Los sistemas de software requieren de mantenimiento para realizar cambios a los mismos, esto implica que se detenga la ejecución del sistema para aplicar las nuevas mejoras. La reflexión permite obtener la información de un sistema en ejecución, lo que da la posibilidad de adaptar el software sin interrupciones. La reflexión del lenguaje Java en cuanto a intercesión se limita únicamente a la modificación de los valores de los campos, por lo que no se ofrece el soporte para manipular los comportamientos del sistema. Este trabajo presenta un esquema de intercesión que hace uso de la información que ofrece la reflexión para aplicar nuevas funcionalidades al sistema.

Palabras clave: Reflexión, adaptación de software, intercesión.

Intercession on Methods' Invocations with Java Reflection

Abstract. Software systems require maintenance to modify them, this involves stopping the system's execution and apply the changes. Reflection allows to get the executing system's information and let adapt the system without any interruption. The intercession in Java programming language is limited only in writing into fields, therefore it is not possible to modify the program's behavior. This paper features an intercession scheme which gets information from reflection to add new functionalities into the existing system.

Keywords: Reflection, adapting software, intercession.

1. Introducción

De acuerdo al estándar 14764 de IEEE [1], los sistemas de software requieren del mantenimiento para añadir soporte a los cambios en el entorno de ejecución, adición de

nuevas funcionalidades y mejoras en las ya existentes. Existen situaciones en las que el código fuente no está disponible o la documentación existente no satisface las cuestiones del equipo de desarrollo. Existen enfoques para modificar el sistema en su representación binaria o en código intermedio, entre ellos están los marcos de trabajo para la manipulación de *bytecodes* como ASM y Javassist; sin embargo, se requiere que los desarrolladores tengan el conocimiento completo de la estructura del sistema; además los cambios se aplican en tiempo de compilación, es decir, es necesario detener el sistema e implica el riesgo de fallas en cualquier parte de la ejecución.

Las capacidades de reflexión permiten obtener la información precisa del sistema en tiempo de ejecución como, por ejemplo, el tipo real de un objeto o acceder a los elementos que están ocultos por sus modificadores. La reflexión del lenguaje de programación en Java limita la capacidad para realizar cambios a una clase únicamente a la modificación de valores de los campos; el uso de *proxies* en Java para la intercesión de métodos sólo es compatible con las interfaces que se especifican durante su construcción y reduce el acceso a otros elementos de una clase concreta.

Los trabajos relacionados presentan enfoques para controlar el uso de la reflexión durante la ejecución de los sistemas y para la implementación de soluciones utilizando dicha técnica.

Este trabajo presenta un esquema de intercesión para la invocación de métodos de un sistema mediante la creación de clases en Java para reemplazar la funcionalidad existente con una alterna; se dispone de AspectJ para interceptar las ejecuciones de cada método y recuperar la información necesaria para aplicar la intercesión.

Este artículo se compone de la siguiente forma: la sección 2 se describen los trabajos relacionados en cuanto a intercesión. En la sección 3 se analizan los fundamentos de reflexión y las herramientas que implementan esta técnica. En la sección 4 se analizan los enfoques de intercesión y sus desventajas. En la sección 5 se presenta y se describe el esquema de intercesión propuesto. En la sección 6 muestra un ejemplo aplicando el esquema de intercesión. En la sección 7 se analizan los resultados. En la sección 8 se dan a conocer las conclusiones y el trabajo a futuro.

2. Trabajos relacionados

En [2] se presentó un análisis estático para los flujos de control implícitos que existen en la reflexión de Java y en los intentos en Android, lo cual permite determinar qué procedimientos se llaman y qué datos se pasan; para el caso de reflexión se analizan las llamadas reflexivas y conocer qué clases se manipulan. En [3] se desarrolló Mirror, una biblioteca para añadir las capacidades de reflexión al lenguaje de programación C++ y de esta forma implementar el patrón de diseño Factory para la generación de nuevas instancias. En [4] se presentó un modelo que contempla cinco dimensiones de control en el meta-nivel que se reportan en otras literaturas relacionadas: control espacial y temporal, control de colocación, control de nivel y control de identidad; este modelo se implementó en el entorno de programación Pharo. En [5] resaltó que aplicar la reflexión hacia los elementos privados de una estructura de datos rompe la encapsulación en objetos; se propuso una política de control que determina si alguna acción reflexiva se considera haber roto dicha encapsulación. En [6] se presentó DroidRA, un enfoque para controlar las llamadas reflexivas en las aplicaciones Android que complementa a las

herramientas de análisis estáticos. En [7] se demostró que la técnica de "cadenas de despachado" trata el problema en el desempeño al aplicar las técnicas de meta-programación, sobre todo las operaciones de reflexión; esta técnica es aplicable tanto para generación de compiladores just-in-time como para las evaluaciones parciales.

3. Reflexión

La reflexión es la capacidad de un programa para analizar y modificar a sí mismo y a su entorno en tiempo de ejecución. La reflexión permite realizar dos acciones: introspección e intercesión. La introspección es la acción de analizar la estructura de un tipo de dato, la composición de estados y comportamientos, y los valores actuales de cada estado; la intercesión permite modificar la estructura del tipo de dato: añadir nuevos estados, modificar su comportamiento y modificar su jerarquía. La información que describe a cada tipo de dato, así como también la de cada elemento que componen al primero, se conocen como meta-objetos; al modificar un meta-objeto sus efectos se reflejan en la ejecución del programa [8].

3.1. Reflexión en Java

El lenguaje de programación Java ofrece la característica de reflexión mediante la importación de clases en el espacio de nombres {java.lang.reflect}, la descripción de cada elemento o meta-objeto se representa como un objeto de tipo del elemento que lo identifica (ej. Method). Por ejemplo, la clase java.lang.Object es una instancia de tipo java.lang.Class, por lo tanto, toda clase que se carga a la máquina virtual de Java es una instancia de tipo Class; el tipo de dato Class, de forma redundante es una instancia de sí mismo, entonces Class es una meta-clase. Desde una instancia de tipo Class, es posible obtener otros meta-objetos en relación, como los campos, los métodos y los constructores que componen a una clase, sin importar el nivel de acceso de cada uno, además permite la creación de instancias detallando la información del meta-objeto del constructor. Desde un meta-objeto de un campo se obtiene la información acerca de sus características, como por ejemplo el tipo de dato y sus modificadores, y permite recuperar o modificar el valor que contiene. Desde un meta-objeto de un método, se obtiene la misma información a la de un campo, y permite la invocación correspondiente. Desde reflexión es posible obtener información que no está disponible en tiempo de compilación como, por ejemplo, conocer el verdadero tipo de dato de una instancia en una variable de tipo Object [9].

3.2. Proxy en Java

La reflexión en Java ofrece la capacidad para crear elementos que intervienen en la interacción entre dos objetos, la generación de *proxies* dinámicos. La clase java.lang.reflect.Proxy cuenta con métodos para la creación tanto de clases como instancias de este tipo en tiempo de ejecución. Para la creación de una clase proxy se hace uso de un cargador de clase y una lista de interfaces; para la generación de instancias se dispone de los dos elementos anteriores y una instancia de tipo InvocationHandler, que contiene la implementación para intervenir en la interacción de

un objeto y realizar las invocaciones al mismo con reflexión. Como resultado, la invocación a un método de un objeto se realizará a través de la instancia InvocationHandler y, posteriormente, al primer elemento; además la instancia de proxy es compatible con los tipos que se especificaron en la lista de interfaces.

3.3. AspectJ

AspectJ es un lenguaje orientado a aspectos que extiende al lenguaje de programación Java para la creación e implementación de aspectos. Se establecen cortes en puntos, que se componen de puntos de unión que identifican en qué partes del código del programa principal se aplicarán los cortes y la funcionalidad del aspecto, esto último mediante el uso de avisos. AspectJ trabaja con los mecanismos de reflexión de Java en lo que respecta a la información dinámica y estática de un punto de unión. Por ejemplo, es posible determinar, por la parte dinámica, el objeto donde se realizó la invocación al método y consultar los valores de la lista de parámetros; mientras que por la parte estática se describe las propiedades del punto de unión desde el *bytecode* [10,11].

4. Intercesión

La intercesión de la reflexión en Java se limita en la modificación en los valores de los campos, no tiene soporte para la manipulación en la estructura de una clase ni en el comportamiento de la misma. Un enfoque para la manipulación del comportamiento en una clase es la implementación de *proxies*; Java ofrece la característica para generar *proxies* en tiempo de ejecución o *proxies* dinámicos, con la capacidad de establecer funcionalidad antes y después de la invocación al método de un objeto. Sin embargo, un objeto proxy sólo se aplica para las interfaces que se especifiquen durante su construcción. El uso de *proxies* implica generar un objeto de este tipo por cada objeto que compone al sistema. Otra solución es la transformación de clases en su representación en *bytecode* mediante el uso de biblioteca, tales como ASM y Javassist; pero estas modificaciones se realizan en tiempo de compilación, lo que requiere detener el sistema para aplicar los nuevos cambios y que los desarrolladores tengan los conocimientos acerca de la composición de cada clase del sistema.

5. Esquema de intercesión propuesto

En esta sección se presenta un esquema de intercesión en métodos en tiempo de ejecución, lo que evita la necesidad de detener el sistema. En la figura 1 se presenta el diagrama de clases del modelo para la intercesión.

El diagrama en la figura 1 muestra la interfaz Intercesión con los métodos intercede() y override() que aplican la ejecución alterna; interrupt() indica si la funcionalidad original se reemplazará con el objeto de intercesión; getDescriptor() devuelve una referencia a un objeto de tipo TargetDescriptor que describe en qué parte del sistema se aplicará la intercesión. En la clase TargetDescriptor se especifica la clase, el nombre del método y la lista de los tipos de los argumentos.

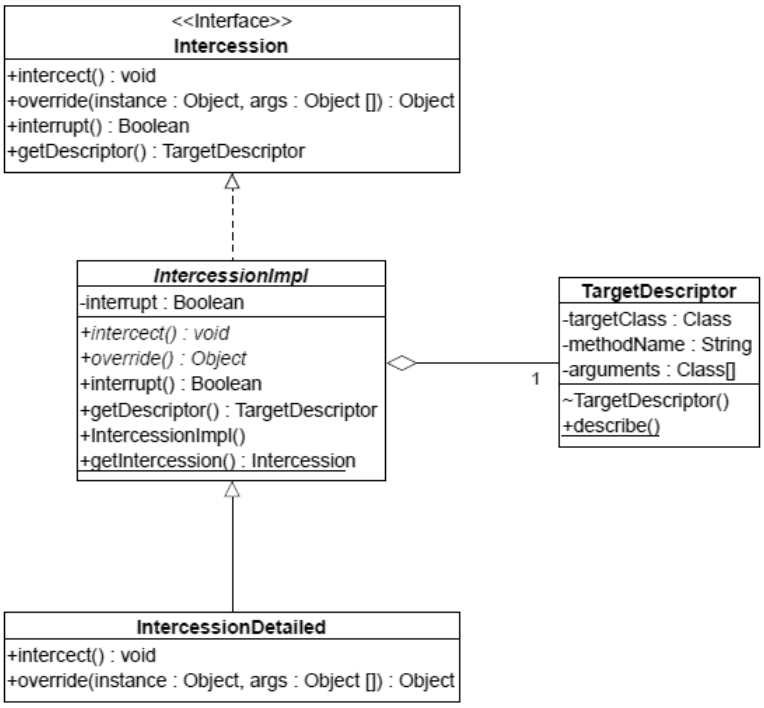


Fig. 1. Diagrama de clase del esquema de intercesión.

```
1  public aspect Intercect {
2      Object around() : execution(!static * Object+.*(..)) {
3          boolean interrupt = false;
4          Intercession intercession = IntercessionImpl.getIntercession(
5                  joinpoint.getTarget().getClass(),
6                  joinpoint.getSignature().getName(),
7                  ((CodeSignature)joinpoint.getSignature()).getParameterTypes()
                       );
8
9          if(intercession != null) {
10             intercession.intercede();
11             interrupt = intercession.interrupt();
12         }
13
14         if(!interrupt) {
15             return proceed();
16         }
17
18         return intercession.override(joinpoint.getTarget(), joinpoint.getArgs
               ());
19     }
20 }
```

Fig. 2. Intercesión de métodos.

La clase IntercesiónImpl es la responsable de recuperar el objeto de intercesión de un conjunto de este tipo. El desarrollador implementa la nueva funcionalidad mediante una clase concreta que redefine los métodos intercede() y override().

En el código Fig. 1, el aspecto Intercect realiza los cortes a todos los métodos de alcance de instancia, su aviso obtiene la información del punto de unión que corresponde a la invocación de un método (línea 2). El aspecto invoca a IntercesiónImpl.getIntercesión() para recuperar un objeto que coincida con el punto de unión (línea 4-7); si el objeto existe (línea 9) y la variable interrupt es verdadera (línea 14), entonces se reemplaza la funcionalidad original por la que se definió en Intercesión.override() (línea 18), si el objeto no existe se procede de forma normal con la ejecución (línea 15).

6. Ejemplo de intercesión

En la sección anterior se mostró el uso de AspectJ para permitir la intercesión mediante clases en Java. El ejemplo (Fig. 2) presenta una clase que simula un dispositivo, el cual reporta el porcentaje de procesamiento y el consumo de memoria.

```
1  public class Machine implements Runnable {
2      private boolean operating = false;
3      private byte cpuUsage;
4      private int memoryUsed;
5      @Override
6      public void run() {
7          turnOn();
8      }
9      public void turnOn() {
10         operating = true;
11         while(operating) {
12             cpuUsage = (byte)((Math.random() * 100) % 100 + 1);
13             memoryUsed = (int)((Math.random() * 1024) % 1024 + 1);
14             Thread.sleep(1000);
15         }
16     }
17     public void turnOff() {
18         if(operating)
19             operating = false;
20     }
21 }
```

Fig. 3. Clase Machine (máquina).

La clase MachineRunIntercesión (Fig. 3) hereda de IntercesiónImp (línea 1) e implementa los métodos intercede() y override() (líneas 5-12); su funcionalidad se aplica en la invocación al método Machine.turnOn() (línea 3) y mostrará en pantalla al usuario un mensaje de que la máquina inició su ejecución (línea 7).

La clase MachineEndedIntercesión (Fig. 4) comparte las mismas características al código anterior. En este caso se reemplazará la funcionalidad del método Machine.turnOff() mostrando un mensaje de que la máquina no detendrá su ejecución (líneas 12-16).

```
1  public class MachineRunIntercession extends IntercessionImpl {
2      public MachineRunIntercession() {
3          super(TargetDescriptor.describe(Machine.class, "turnOn", new Class
               [0]));
4      }
5      @Override
6      public void intercede() throws RuntimeException {
7          JOptionPane.showMessageDialog(null, "This machine has started on " +
               Thread.currentThread().toString());
8      }
9      @Override
10     public Object override(Object instanceSource, Object[] args) throws
           RuntimeException {
11         return null;
12     }
13 }
```

Fig. 4. Clase de intercesión al invocar el método Machine.turnOn().

```
1  public class MachineEndedIntercession extends IntercessionImpl {
2      public MachineEndedIntercession() {
3          super(TargetDescriptor.describe(Machine.class, "turnOff", new Class
               [0]));
4          super.interrupt = false;
5      }
6
7      @Override
8      public void intercede() throws RuntimeException {
9          JOptionPane.showMessageDialog(null, "This has ended.");
10     }
11
12     @Override
13     public Object override(Object instanceSource, Object[] args) throws
           RuntimeException {
14         JOptionPane.showMessageDialog(null, "Lies. This will continue.");
15         return null;
16     }
17 }
```

Fig. 5. Clase de intercesión al invocar el método Machine.turnOff().

```
1  public class DisplayFahrenheit extends reflect.IntercessionImpl {
2      public DisplayFahrenheit() {
3          super(TargetDescriptor.describe(weather.util.display.DisplayManager.
               class, "displayTemperature", new Class[] { int.class }));
4      }
5      public void intercede() throws RuntimeException {
6          super.interrupt = true;
7      }
8      public Object override(Object instanceSource, Object[] args) throws
           RuntimeException {
9          int value = (int)args[0];
10         int result = (int)(((float)value * (9f / 5f)) + 32f);
11         ui.getTxtTemp().setText(Integer.toString(result) + " F");
12         return null;
13     }
14 }
```

Fig. 6. Clase para representar la temperatura en grados Fahrenheit.

7. Resultados

El esquema de intercesión se aplicó en un sistema de reporte de las condiciones del clima en tiempo real, para representar y visualizar los datos en otras unidades de medición. Se generaron las clases para reemplazar la ejecución original. AspectJ realizó los cortes en todos los métodos del sistema, se analizó con la reflexión la información de cada método para determinar la existencia alguna ejecución alterna. Por ejemplo, el código 5 (Fig. 5) contiene la ejecución para representar la temperatura en grados Fahrenheit; reemplazando la definición original del método intercesión displayTemperature() de la clase intercesión DisplayManager.

Este esquema permitió modificar el sistema sin detenerlo lo que evitó la pérdida de la información, a diferencia de aplicar los cambios en tiempo de compilación.

8. Conclusión y trabajo a futuro

Se presentó un esquema de intercesión para las invocaciones de métodos sin detener la ejecución del programa y se implementó en un sistema de reporte de las condiciones del clima en tiempo real, para representar y visualizar los datos en otras unidades de medición; se aplicó la reflexión para obtener la información de la invocación en cada método y se trabajó con AspectJ para aplicar los cortes en todos los métodos del sistema.

El uso de la reflexión permite obtener la información del sistema que no está disponible en tiempo de compilación, lo que da la posibilidad de realizar adaptaciones seguras sin afectar en forma negativa la integración del sistema.

Como trabajo a futuro se considera trabajar con los cargadores de clase dinámicos para permitir la carga e intercambio de clases que implementen intercesiones con el propósito de realizar el mantenimiento al sistema en tiempo de ejecución.

Agradecimientos. Este trabajo cuenta con apoyo por parte del Consejo Nacional de Ciencia y Tecnología (CONACYT).

Referencias

1. IEEE, ISO/IEC: Norma IEEE Std 14764-2006 Revision of IEEE Std 1219-1998, IEEE std, IEEE (2006)
2. Barros, P., Just, R., Millstein, S., Vines, P., Dietl, W., d'Amorim, M., Ernst, M.D.: Static analysis of implicit control flow: Resolving Java reflection and Android intents. In: ASE 2015: Proceedings of the 30th Annual International Conference on Automated Software Engineering, Lincoln, NE, USA, pp. 669–679 (2015)
3. Chochlik, M.: Implementing the factory pattern with the help of reflection. Computing and Informatics (2015)
4. Papoulias, N., Denker, M., Ducasse, S., Fabresse, L.: Reifying the reectogram: Towards explicit control for implicit reflection. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, SAC '15, New York, NY, USA, ACM, pp. 1978–1985 (2015)

5. Teruel, C., Ducasse, S., Cassou, D., Denker, M.: Access control to reflection with object ownership. SIGPLAN Not, 51(2), pp. 168–176 (2015)
6. Li, L., Bissyandé, T. F., Octeau, D., Klein, J.: Droidra: Taming reflection to support whole-program analysis of android apps. In: Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, New York, NY, USA, ACM, pp. 318–329 (2016)
7. De Wael, M., Marr, S., Van Cutsem, T.: Fork/join parallelism in the wild: Documenting patterns and anti-patterns in java programs using the fork/join framework. In: Proceedings of the 2014 International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools, PPPJ '14, pp. 39–50 (2014)
8. Forman, I. R., Forman, N.: Java Reflection in Action (In Action Series). Manning Publications Co., Greenwich, CT, USA (2004)
9. Oracle-Corporation: Trail: The reflection API. https://docs.oracle.com/javase/tutorial/reflect/
10. Eclipse-Foundation: Aspectj documentation. https://eclipse.org/aspectj/docs.php
11. Laddad, R.: AspectJ in Action: Practical Aspect-Oriented Programming. Manning Publications Co., Greenwich, CT, USA (2003)

ISSN 1870-4069

Mixing Scrum-PSP: Combinación de Scrum y PSP para mejorar la calidad del proceso de software

Mauricio Leonardo Urbina Delgadillo, María Antonieta Abud Figueroa, Gustavo Peláez Camarena, Giner Alor Hernández, Alma Ivonne Sánchez García

División de Estudios de Posgrado e Investigación, Instituto Tecnológico de Orizaba, Veracruz, México

mauricio.urbina.isc@gmail.com, mabud@ito-depi.edu.mx, sgpelaez@yahoo.com.mx, galor@itorizaba.edu.mx, alivsaga@hotmail.com

Abstract. El desarrollo de software en un entorno real es muy agresivo con respecto a los tiempos de entrega, presupuesto y el contrato con el cliente. La combinación de métodos ágiles y procesos disciplinados es una opción prometedora que apoya al desarrollo de software a mejorar la calidad. En este trabajo se presenta un modelo de integración de procesos combinando Scrum y PSP (Personal Software Process) para mejorar el proceso de software, teniendo en cuenta las características primordiales de cada marco de trabajo. El resultado de esta investigación es un intento para adoptar la agilidad y disciplina de los procesos en diferentes ambientes de trabajo.

Keywords: Calidad de software, gestión del proceso de software, mejora de proceso de software, proceso personal de software, PSP, scrum.

Mixing Scrum-PSP: Combining Scrum and PSP to Improve Software Quality Process

Abstract. Software development into the enterprise environment is characterize by aggressive delivery times, low budget and customer contract. Combination of agile methods and process disciplines is a suitable option for the software development process to improve the quality. In this paper we show a proposal for integrate a process model between Scrum and Personal Software Process to improve the quality of the development process, considering the main characteristics of each approach. The result of this research is an attempt to adopt the agility and discipline in different work environments.

Keywords: software quality, software process management, software process improvement, personal software process, PSP, Scrum.

1. Introducción

El desarrollo de software en un entorno de proyecto real se caracteriza por ser rápido, cambiante y complejo. Debido a que los problemas requieren soluciones especializadas, los ingenieros de software se enfrentan a calendarios agresivos y realistas para generar un producto de calidad sobresaliente. Otro factor que se une es la impaciencia por terminar en el plazo acordado, no exceder el presupuesto y cumplir con los requisitos del cliente merman la calidad del producto final.

Gracias a los métodos ágiles como SCRUM, XP, Dynamic System Development Method (DSDM), Crystal, Lean Development (LD), los equipos de desarrollo de software realizan entregas en poco tiempo y los proyectos se adaptan a las condiciones cambiantes del cliente. Por otra parte, los modelos de calidad para la administración del proceso de desarrollo como CMMI-DEV, SPICE, Six Sigma y PSP/TSP permiten tener una buena gestión del proceso utilizando guías para supervisar que los desarrolladores sigan el flujo establecido por la empresa y ayudan a mejorar calidad del producto final.

Los métodos agiles y los procesos disciplinados representan casos de extremo a extremo en sus paradigmas, y siempre habrá pequeñas discordias entre ambos por parte de los desarrolladores de software [1]. Sin embargo, los dos enfoques tienen características para complementarse uno al otro. Las grandes compañías y los grandes proyectos de software utilizan ambos enfoques en la medida de sus objetivos y su entorno.

Uno de los métodos agiles más populares es Scrum, considerado como un marco de trabajo [2] [3] formado por un conjunto de prácticas y reglas, el desarrollo del software se realiza de manera iterativa e incremental en flujos de trabajo rápidos donde se construye un incremento funcional del sistema. Del lado de los procesos disciplinados se encuentra PSP^{SM}, conocido un marco de trabajo estructurado de formularios, normas, y procedimientos para el desarrollo de software [4], su propósito es ayudar a mejora de las habilidades de ingeniería de software a los desarrolladores.

Esta investigación está enfocada en la relación entre Scrum y PSP para mejorar la calidad del software. Por esta razón se plantea la opción de combinar Scrum y PSP para mejorar el proceso para desarrollo de software en diferentes entornos. Este documento se estructura de la siguiente manera: en la Sección 2 se presentan a Scrum y PSP. En la Sección 3 se mencionan trabajos relacionados con la investigación. La Sección 4 presenta la integración entre Scrum y PSP. El diseño del caso de estudio se encuentra en la Sección 5. Las conclusiones y las líneas de trabajo futuro se describen en la Sección 6.

2. Scrum y PSP

2.1. Scrum

Jeff Sutherland aplicó los principios al desarrollo de software en 1993 en Easel Corporation y no fue hasta 1996 junto con Ken Schwaber que se presentaron como válidos para gestionar el desarrollo de software [3]. Es definido como un marco de trabajo por el cual las personas pueden trabajar con problemas complejos adaptativos, al mismo tiempo se entregan productos de gran valor posible. Caracterizado por ser

ligero, fácil, y difícil de dominar; está más orientado a las personas que a los procesos porque la gestión no se basa en el seguimiento de un plan sino en la adaptación continua a las circunstancias de un proyecto.

El ciclo de vida de Scrum se compone de tres fases *Pre-juego*, *Juego* y *Post-juego* (Figura 1). Scrum incluye tres características consideradas críticas para la implementación del control de procesos empíricos [5]:

- ***Transparencia:*** los procesos significativos deben ser visibles para aquellos que son responsables del resultado. Se requiere que dichos aspectos sean definidos por un estándar común, de tal modo que los involucrados compartan un conocimiento general de lo que está viendo.
- ***Inspección:*** Los involucrados tienen que inspeccionar con frecuencia los artefactos generados y el progreso hacia el objetivo del "sprint" para detectar variaciones indeseables. Estas inspecciones no tienen que ser muy frecuentes para que no interrumpa el ciclo de trabajo.
- ***Adaptación:*** Si la inspección determina que uno o más aspectos de un proceso se alejan de los límites aceptables, y que el producto resultante no será aceptado, el proceso o el material que se está siendo trabajando tiene que ser ajustado. El ajuste debe realizarse para minimizar desviaciones mayores.

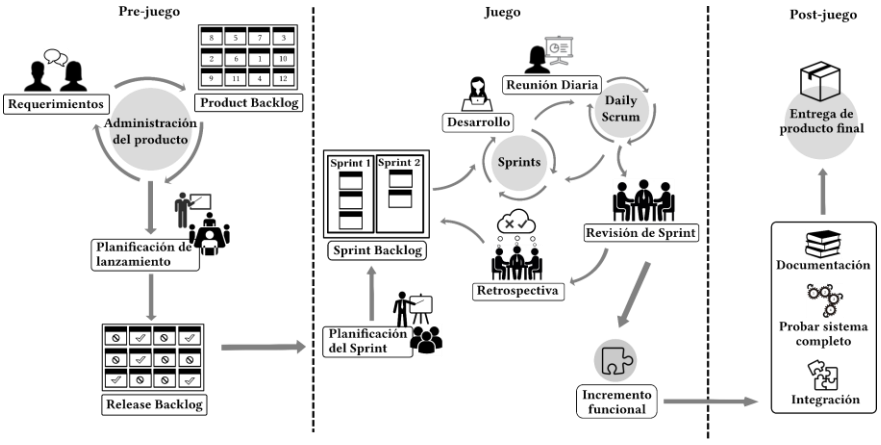


Fig. 1. Ciclo de vida de Scrum.

2.2. Personal Software Process (PSP)

PSP en un marco de trabajo para mejorar los procesos de construcción de software a un nivel personal, su estrategia permite mejorar el rendimiento de los programa-dores utilizando prácticas sólidas para monitorizar y esforzarse en mejorar el desempeño, y la calidad de los productos. PSP se basa en los siguientes principios de planificación y calidad [4]:

- Cada desarrollador es diferente, los desarrolladores tienen que planear su trabajo con base a su información personal.

- Para una mejorar continua del desempeño, los desarrolladores definen y miden correctamente sus procesos.
- Para producir productos de calidad, los desarrolladores deben sentirse responsables de la calidad de su trabajo.
- El costo es menor si se encuentran y reparan defectos en fases tempranas que en fases próximas.
- Es más eficiente prevenir los defectos que buscarlos y repararlos.
- El camino correcto el siempre el más rápido y barato de trabajar.

PSP define siete niveles de madurez (Figura 2), los niveles PSP0 y PSP1 son críticos para aprender la disciplina del proceso, considerados punto de partida para la secuencia del proceso. Cada nivel de PSP se dividen en fases o procesos, descrita por medio de un *Scripts* el cual no solo especifica los pasos a seguir también los criterios de entrada y salida. Humphrey se centra en tres fases: *diseño*, *código* y *pruebas* pues son las actividades que más tiempo consumen en el desarrollo del software.

3. Trabajos Relacionados

En este apartado se presentan investigaciones de varios autores realizadas con un objetivo similar entre ellas, la combinación de métodos agiles y métodos disciplinados para mejorar la ingeniería de software en factores como la gestión de procesos y administración de la calidad.

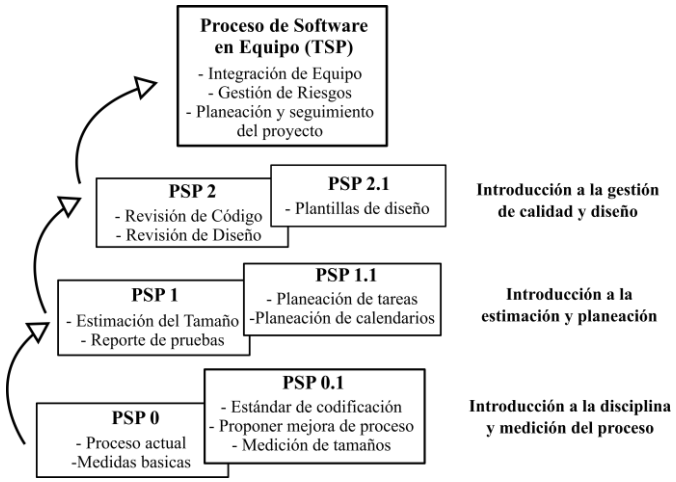


Fig. 2. Niveles de Madurez de PSP (Fuente: PSP: A Self-Improvment Process for Software Engi-neers, p. 8).

A continuación se presenta un cuadro comparativo (Tabla 1) que permite una rápida y clara perspectiva de trabajo realizado en las investigaciones, la primera columna muestra los métodos utilizados por los autores, la segunda describe el análisis utilizado en la investigación, la tercera muestra brevemente la combinación resultante.

Tabla 1. Análisis comparativo entre trabajos anteriores relacionados con la modificación entre métodos ágiles y métodos disciplinados.

Métodos utilizados	Diseño metodológico	Combinación resultante
Boehm,2002 [6]		
Agiles: XP, Adaptive Software Development, Crystal, DSDM. *Disciplinados:* basados en riesgos, basados en planes, CMM, CMMI.	Comparación entre 7 áreas clave para cada enfoque (*Desarrolladores, Clientes, Requerimientos, Arquitectura, Ajustes, Tamaño* y *Objetivo principal*) en donde cada método tiene ventajas en determinados proyectos.	Por sugerencia del autor, realizar análisis de riesgos basado en las características del proyecto utilizando las 7 áreas descritas para determinar un balance entre los enfoques.
Paulk, 2002[7]		
Agiles: Principios del manifestó ágil y XP. *Disciplinados:* CMM y SW-CMM.	Comparación en contra entre las características: *Velocidad / Disciplina, Individuos / Procesos, Desarrollo / Documentación, Respuesta a cambios / Planeación, Colaboración de Clientes / Contratos*.	Por sugerencia del autor, recolectar información sobre el tipo de proyecto, generar un análisis considerando la naturaleza del proyecto para la integración de prácticas de ambos enfoques.
Sutherland, 2007[8]		
Ágil: Scrum y Lean. *Disciplinado:* CMMI nivel 2 y 3.	A través de 12 Practicas Genéricas asociadas con CMMI se establece la disciplina dentro de los proyecto de una empresa. Scrum ayuda a resolver problemas de CMMI.	Combinación entre CMMI, Scrum y Lean. Ofrece disciplina y agilidad para la institucionalización de procesos dentro de una empresa.
Jerzy Nawrocki, 2001 [9]		
Ágil: XP. *Disciplinados:* CMMI.	Estableciendo un análisis para relacionar diferentes prácticas de XP con Prácticas Especificas de CMMI nivel 2.	eXtrem Programming Maturity Model (XPMM) de cuatro niveles: *No cumple con nada, Inicial, Avanzado, Maduro*. Un proceso incremental definido por un conjunto de prácticas obligatorias de XP para cada nivel del proceso.
Yani Dzhurov, 2009 [10]		
Agiles: XP. *Disciplinados:* PSP.	Modificación de PSP para aligerar y hacer más fácil el proceso de desarrollo manteniendo sus principios de básicos. Con 6 prácticas de PSP y 6 prácticas de XP se estructuró una metodología que apunta hacia una mejor planeación y control de la calidad del desarrollo de software.	Personal eXtrem Programming (PXP) es un proceso iterativo y comprende un par de iteraciones y ciclos controlados. Conformado por siete fases: *Requerimientos, Planificación, Iteración inicial, Diseño, Implementación, Pruebas de Sistema, Retrospectiva*.

Métodos utilizados	Diseño metodológico	Combinación resultante
Brown, 2014 [11]		
Agiles: Scrum. *Disciplinados:* PSP y SEMAT.	Basado en un análisis de las adaptaciones Scrum-PSP y PXP se propuso una adaptación que reutilice las prácticas de PSP en cualquier método de desarrollo de software. Se utilizó la notación de SEMAT para expresar en un lenguaje común las prácticas de PSP.	Integración con el núcleo SEMAT, muestra a PSP como una alpha "*way of working*". Los 7 niveles de PSP son representados como estados de un sub-alpha "*PSP Compliance*". Igualmente se adapta Scrum utilizando el Backlog, el incremento y el Sprint como alphas dentro de SEMAT.

4. Mixing Scrum-PSP: agilidad y disciplina trabajando juntos

En esta sección se presenta la integración de los dos métodos, Scrum y Personal Process Software (PSP), como primer punto se muestra un análisis de la compatibilidad que existe entre ambos enfoques basado en sus características, puntos fuertes y debilidades; por último se describe la integración resultante.

4.1. Análisis de compatibilidad

Este análisis tiene como propósito generar una comparación para determinar las características de integración y las actividades que sirven para mejorar el proceso de desarrollo del software (Tabla 2).

Una brecha encontrada en Scrum es la carencia de pautas para el proceso de desarrollo que orienten al desarrollador en la construcción del incremento, aquí es donde PSP brinda una serie de prácticas sencillas para la construcción del incremento. Además, gracias a la recolección de datos, por parte de PSP, genera registros donde la información es utilizada para predecir un mejor tiempo de trabajo en las actividades.

La esencia de Scrum son: la agilidad, adaptación a cambios, la estimación del tiempo de trabajo y el proceso iterativo e incremental. Estas características son consideradas parte fundamental del esqueleto de la integración. Por otro lado, para tener una base sólida existen 5 expectativas de PSP, mencionadas por Humphrey [3], que deben cumplirse en cualquier implementación: La definición las métricas y/o características de calidad que se aplicarán al proyecto y su producto, Cumplir con la recolección de los datos de todas las actividades realizadas en los proyectos para generar la base de conocimientos que permitirá hacer las estimaciones a futuro, Obligatorio que todos los integrantes del equipo dispongan de los recursos con libertad y facilidad que les ayuden a realizar sus actividades, Cada integrante del equipo tiene que seguir el flujo del proceso con disciplina, Realizar la medición del producto final y por ende que cumpla con la meta establecida.

Tabla 2. Análisis comparativo entre factores de Scrum y PSP.

Factores	Scrum	PSP
Enfoque	Agilidad en desarrollo. Adaptación a cambios e Inspecciones del trabajo. Producción rápida.	Mejorar las habilidades personales del desarrollador. Estimación basada en datos históricos. Predictibilidad.
Conocimiento	Empírico, Tácito.	Teórico basado en registros.
Prácticas	Gestión del proceso centrado en los requerimientos del cliente. Priorización de los requerimientos y estimación del tiempo. Descomposición de tareas.	Establece un flujo de trabajo para el ingeniero de manera personal. Define guías (scripts) para la administración del proceso.
Ciclo de vida	Iterativo e incremental. Define tres fases: Pre-juego, Juego y Post-juego.	Iterativo y escalonado por niveles. Define tres principales procesos: Planificación, Desarrollo y Postmortem.
Ambiente	Cambiante, inquieto, rápido, enfocado al proyectó.	Estable, pocas modificaciones, enfocado a la disciplina.
Cultura	Trabajo y colaboración en equipo. Permite una mejora continua para todos los integrantes del equipo.	Establece la disciplina y el respeto al proceso de trabajo. Establece una mejora continua de las habilidades personales del desarrollador.

4.2. Mixing Scrum-PSP (MSP)

La integración de Scrum con PSP proporciona pautas para los integrantes del equipo de desarrollo en mejorar la administración del proceso con simples prácticas. *MSP* está formado por dos capas de procesos: el *Ciclo de Vida MSP* y La *Iteración de Desarrollo*.

Ciclo de Vida MSP. Definido por las tres fases *Preparación*, *Desarrollo* y *Entrega* (Figura 3), las fases de planeación y entrega son compaginadas con las fases de *Pre-juego* y *Post-juego* de Scrum, se cambiaron los nombres para una mejor identificación por parte de los involucrados.

La fase de *Preparación:* Abarca la definición del proyecto, la obtención de las necesidades del cliente que posteriormente se convertirán en requisitos del sistema y la creación del *Product Backlog*. La planificación estratégica de recursos es conforme al *Product Backlog* para determinar un calendario de entregas y estimación de tiempos para el proyecto, también se realiza un plan de desarrollo y análisis de riesgos. Conforme los requisitos son priorizados por el equipo de desarrollo se seleccionan los que componen el *Backlog de Liberación*. Las estimaciones estarán apoyadas por el método estadístico *PROBE* de PSP, el cual tiene buena precisión para calcular el costo de las actividades en esfuerzo y tiempo, según los informes del *Software Engineering Institute* (SEI).

La fase de *Desarrollo*: El equipo de desarrollo elige cuales son las actividades del *Backlog de Liberación* que entrarán el *Sprint Backlog* y el número de sprint utilizados para desarrollar todos los requerimientos. Se integran las prácticas de PSP basadas en las tres principales fases: *Planeación*, *Desarrollo* y *Postmortem*. La revisión diaria

(*Scrum Daily*) y la revisión de Sprint continua siendo una actividad sin cambios provenientes de Scrum, de esto modo se permite para detectar y remover posibles contratiempos que impidan el avance normal del proyecto. La Retrospectiva estará apoya por las fase de *Postmortem* de PSP

La fase de *Entrega*: Establece la integración del nuevo incremento con el incremento generado en el anterior *Sprint*. Se realizan las pruebas, que establezca la organización, a la liberación para verificar su funcionamiento en diferentes entornos. Un punto importante de esta fase es que la documentación del proyecto tiene que estar terminada y verificada.

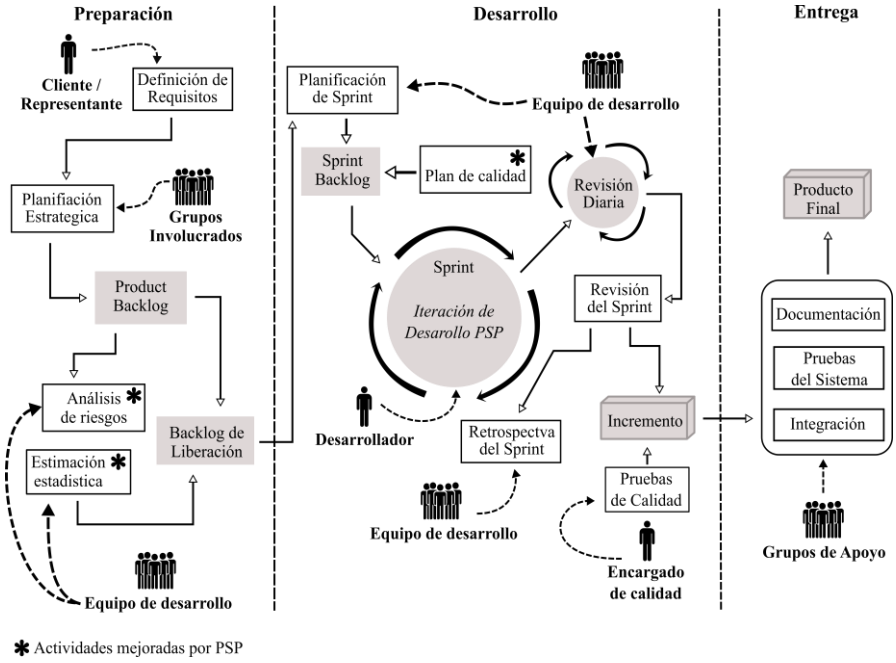


Fig. 3. Ciclo de vida de Mixing Scrum-PSP.

Iteración de Desarrollo. Es el proceso en el cual cada desarrollador utiliza las actividades clásicas para construir los programas o módulos correspondientes a sus actividades asignadas, ubicada en el *Sprint*. El marco de trabajo de PSP se centra en tres procesos: *diseño*, *código* y *pruebas*; pues son las actividades que más tiempo consumen del desarrollo del software [3]. El flujo de trabajo (Figura 4) está basado en: PSP 2.1, la iteración consiste en 8 actividades descritas a continuación:

- *Planificación:* Se produce un plan detallado para trabajar el desarrollo del programa definido por los requisitos del problema, los formatos para escribir el plan de trabajo no son difíciles pero requieren toda la atención del desarrollador. El plan consiste en la obtención y definición de los requerimientos para el programa escritos en documento claramente y sin ambigüedades, y una buena estimación de tiempo para completar el desarrollo del programa, apoyado por el método *PROBE*.

- *Diseño Detallado:* Se realiza un diseño detallado para las especificaciones del programa definido por los requerimientos, las herramientas utilizadas es responsabilidad del desarrollador.
- *Revisión de Diseño:* Validación de la consistencia del diseño, estrategia de verificación y detección de errores.
- *Código:* La transformación del diseño a sentencias de lenguajes de programación.
- *Revisión de Código:* Examinar la calidad del código mediante la detección temprana de errores.
- *Compilación:* Se traducen las sentencias del lenguaje de programación a código ejecutable. La mayoría de los defectos de sintaxis serán removidos durante esta fase. Esta *fase* es *opcional*, determinada por el entorno de desarrollo y el lenguaje de programación.
- *Pruebas Unitarias:* Cada desarrollador realiza pruebas unitarias al programa o módulo para verificar que cumpla con los requerimientos, no se establece un número límite para las pruebas o herramientas para realizarlas, sin embargo se tienen que registrar el tipo de cada prueba realizada.
- *Postmortem:* Puede considerarse una retrospectiva personal para resumir y analizar los datos generados por el proceso. Estos datos incluyen valores sobre el tiempo estimado y el tiempo real utilizado, calidad y productividad. Además, la información proporcionada por los productos personales permite dar bases a las retrospectivas del *Sprint*.

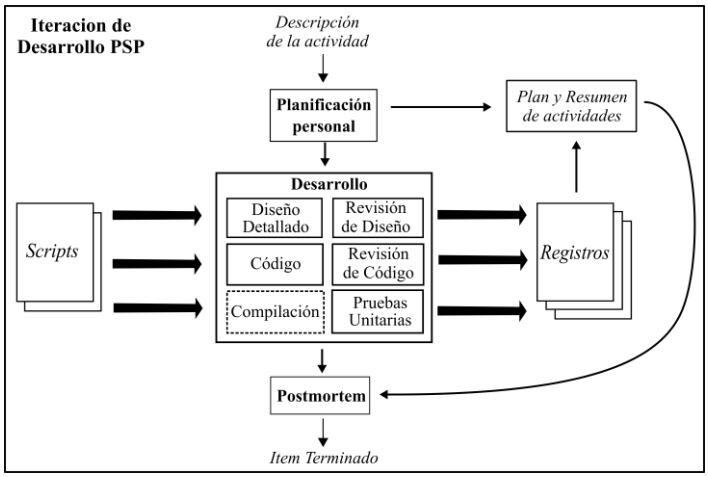


Fig. 4. Iteración de trabajo PSP.

5. Caso de Estudio

Para verificar el modelo de integración y su factibilidad se establece un caso de estudio con el propósito de implementar y documentar la experiencia de combinar dos enfoques para el desarrollo de software, Scrum y PSP, con la finalidad de mejorar la calidad del proceso de desarrollo a través de la introducción de actividades disciplinarias para mejorar las habilidades

de los desarrolladores de software. Por el cual se realiza la siguiente pregunta de reflexión: *¿En qué medida la combinación de Scrum y PSP mejora la calidad del producto final de manera indirecta mejorando la gestión del proceso de construcción?*

Se cuenta con un equipo de desarrollo conformado por tres integrantes dentro de una empresa dedicada al desarrollo de software de telemática, todos tienen conocimientos básicos sobre el método de desarrollo Scrum. La capacitación se realizará por medio de video-tutoriales alojados en una plataforma para aprendizaje en línea, tomando como referencia los materiales proporcionados por el *SEI*.

Para la recolección de datos se utilizarán las herramientas *CONFLUENCE* y *JIRA*, ambas de Altlassian, para la gestión de requerimientos y la gestión de proyectos respectivamente, estas dos herramientas soportan el desarrollo ágil de Scrum. Para la recolección de los datos de PSP se tiene el uso de métodos cuantitativos con énfasis en conocer el rendimiento personal de desarrollador proporcionados por la herramienta *PSP Dashboard*, de Tuma Solutions.

6. Conclusiones y trabajo futuro

En esta investigación se ha presentado una integración de Scrum con PSP preservando la esencia de ambos enfoques que mejora el trabajo individual de los integrantes del equipo introducción nuevas prácticas que fomentan la disciplina del proceso y la institucionalización de mejores prácticas trabajo de la empresa para el desarrollo de software. Una recomendación para la Comunidad Ágil es la extensión de métodos ágiles inspirados en las metas y expectativas que proporcionen los métodos disciplinados que prometen reforzar las debilidades de ambos enfoques.

Como trabajo futuro para esta investigación se debe realizar la comparación de los resultados de implementación con otros métodos de desarrollo de software para crear una perspectiva de los beneficios y debilidades que ofrece el modelo. Mejorar la integración estableciendo un análisis de riesgos similar a [12] para generar un modelo sólido y personalizado para cada proyecto. Enriquecer el modelo con la adicción de normas de calidad como: *ISO/IEC*, niveles de *CMMI-DEV*, prácticas de *MoProSoft* o *CompetiSoft*. Enriquecer la dinámica del trabajo en equipo con prácticas de *Team Software Process* (TSP).

Agradecimientos. Esta investigación fue apoya por El Consejo Nacional de Ciencia y Tecnología (CONACYT) bajo el Programa Nacional de Posgrados de Calidad (PNPC) y el Instituto Tecnológico de Orizaba (ITO) División de Estudios de Posgrado e Investigación (DEPI).

Referencias

1. Boehm, B., Turner, R.: Balancing agility and discipline. Addison-Wesley, Boston (2004).
2. Palacio, J., Ruata, C.: Scrum Manager Gestión de Proyectos. SafeCreative, 4th ed., Disponible en: http://www.safecreative.org/work/1012268137397, pp. 57–87 (2010)
3. Palacio, J.: Flexibilidad con Scrum. 2nd ed., Disponible en: http:// www.safecreative.org/work/0710210187520, pp. 87, 125–176 (2007)
4. Humphrey, W.: PSP: A Self-Improvment Process for Software Engineers. Upper Saddle River, NJ, Addison-Wesley (2005)

5. Sutherland, J., Schwaber, K.: The Definitive Guide to Scrum: The Rules of the Game. Scrum, Org and Scrum Inc. (2014)
6. Boehm, B.: Get ready for agile methods, with care. Computer, 35(1), pp. 64–69 (2002)
7. Paulk, M.: Agile Methodologies and Process Discipline. CrossTalk: The Journal of Defense Software Engineering, 15(10), pp. 15–18 (2002)
8. Sutherland, J.: Scrum and CMMI Level 5: The Magic Potion for Code Warriors. AGILE (2007)
9. Nawrocki, J.: Toward maturity model for extreme programming. Proceedings 27th EUROMICRO Conference, A Net Odyssey (2001)
10. Dzhurov, Y.: Personal Extreme Programming – An Agile Process for Autonomous Developers. In: Proceedings of the International Conference on Software, Services & Sematic Technologies (2009)
11. Brown, D.: PSP Implementations for agile methods: a SEMAT-based approach. Software Engineering: Methods, Modeling, and Teaching, 3, pp. 41–45 (2014)
12. Boehm, B., Turner, R.: Using risk to balance agile and plan- driven methods. Computer, 36(6), pp. 57–66 (2003)

ISSN 1870-4069

Análisis comparativo de patrones de diseño de interfaz de usuario para el desarrollo de aplicaciones educativas

Cesar Augusto Cortes-Camarillo, Giner Alor-Hernández,
Beatriz Alejandra Olivares-Zepahua, Lisbeth Rodríguez-Mazahua,
Silvestre Gustavo Peláez-Camarena

Instituto Tecnológico de Orizaba, División de Estudios de Posgrado e Investigación, México

ccortescamarillo@acm.org, {galor,lrodriguez}@itorizaba.edu.mx,
{bolivares,gpelaez}@ito-depi.edu.mx

Resumen. En la actualidad las tabletas, teléfonos inteligentes y televisores inteligentes se encuentran al alcance de la mayoría de personas, estos dispositivos tienen diversos usos, por esta razón diseñar aplicaciones se ha vuelto una tarea tediosa y difícil de realizar, debido a los diferentes dispositivos y plataformas existentes en el mercado. Los patrones de diseño de interfaz de usuario permiten facilitar el desarrollo de aplicaciones, debido a que cada plataforma tiene su propia interacción entre el usuario y el dispositivo, porque dependiendo del dispositivo se utiliza un patrón de diseño distinto para su interfaz de usuario. Por esta razón se propone un análisis comparativo de patrones de diseño de interfaz de usuario, para generar contextos de uso orientados al ámbito educativo. Se diseñaron contextos de uso para establecer qué patrones de diseño de interfaz de usuario son recomendables para un dispositivo en específico, facilitando el desarrollo de aplicaciones educativas multi-dispositivos.

Palabras clave: Ámbito educativo, contexto de uso, interfaz de usuario, multi-dispositivo, patrones de diseño.

Comparative Analysis of User Interface Design Patterns for Developing Educational Applications

Abstract. Actually, most people have at their disposal different devices including tablets, smartphones and smart TVs, for this reason the design of this kind of applications has become a tedious and difficult task to do, by the different devices and platforms that exist in the market. The UIDPs (User Interface Design Patterns) facilitate the development of the kind of applications by, considering that on each platform the user interacts differently with the device because UIDPs

are used depending on the device. Therefore, a comparative analysis of the UIDPs is proposed to generate contexts of use oriented to the educational field. Contexts of use were designed to establish which UIDPs are suitable for a specific device in order to facilitate the development of multi-device educational applications.

Keywords: Context of use, design patterns, educational context, multi-device, user interface.

1. Introducción

El desarrollo de software es un proceso complicado que involucra la interacción entre usuarios y diseñadores, entre usuarios y herramientas, y entre diseñadores y herramientas. La ingeniería de software se utiliza para establecer mecanismos para el desarrollo de software a través de modelos, metodologías o patrones [1].

Durante el desarrollo del software es común implementar o desarrollar una solución desde cero, esto representa una pérdida de dinero y energía. Los patrones de diseño son una solución probada a un problema en específico. C. Alexander [2] definió al patrón de diseño como "una regla de tres partes, que expresa una relación entre cierto contexto, un problema y una solución". Diferentes áreas de la ingeniería ocupan los patrones de diseño y el desarrollo de software no es la excepción.

El desarrollo del software y el ámbito educativo se encuentran en constante evolución. En la actualidad la tecnología es una parte integral la vida diaria y cada vez es más fácil interactuar con ella, un ejemplo son los dispositivos móviles y la televisión inteligente que se utilizan para el entretenimiento en lugar del ámbito educativo, esto genera que se desaproveche su uso para incrementar el aprendizaje y la enseñanza.

Los patrones de diseño de interfaz de usuario son aquellas soluciones para construir interfaces hombre-máquina a través de una interfaz gráfica. Los patrones de diseño de interfaz de usuario son importantes, pero hace falta un análisis que permita identificar cuáles son los más apropiados para el desarrollo de software en el ámbito educativo. Por esta razón en este trabajo se presenta un análisis de patrones de diseño de interfaz de usuario para generar contextos de uso. En este análisis se utilizan los contextos de uso propuestos por J. Engel et al. [3], pero orientados al ámbito educativo con el objetivo de identificar qué patrón de interfaz de usuario es más conveniente dependiendo del dispositivo en el cual se desarrollará. Entre los dispositivos que se tienen contemplados se encuentran los teléfonos inteligentes, tabletas y televisiones inteligentes.

Este artículo se estructura de la siguiente forma: En la sección 2 se presenta un conjunto de trabajos de investigación acerca de patrones de diseño, contexto de uso de patrones de diseño y aplicaciones para televisiones inteligentes. En la sección 3 se presenta un análisis comparativo de un conjunto de patrones de diseños de interfaz de usuario. En la sección 4 se presenta un caso de estudio donde se utilizan los patrones de uso de interfaz de usuario más importantes para el ámbito educativo. Finalmente, en la sección 5 se presentan las conclusiones y se enfatiza el trabajo a futuro.

2. Estado del arte

Los patrones de diseño de interfaz de usuario se estudiaron en diversos dominios y tipos de aplicaciones. En esta sección se presentan trabajos relacionados clasificados en patrones de diseño de interfaz de usuario y su uso en diversos dominios y tipos de aplicaciones.

2.1. Patrones de diseño de interfaz de usuario

En esta sección se presentan diversos trabajos donde se menciona la importancia de utilizar patrones de diseño de interfaz de usuario. J. Richard et al. [4] presentaron un trabajo sobre la relevancia de utilizar los patrones de diseño de interfaz de usuario, debido a los problemas que enfrentan los desarrolladores al usarlos, sin embargo no se ocupan y esto provoca una pérdida de productividad y calidad. S. Kim [5] mencionó que existen muchas actividades de investigación para la ingeniería basada en modelos de interfaces de usuario para múltiples dispositivos. Se identificaron tres principales limitaciones para los enfoques convencionales y se propuso un marco de trabajo llamado PELUM (*Pattern and Event based Logical UI Modeling*, Patrones y Eventos Basados en Modelado de la Interfaz de Usuario) para modelar interfaces de usuario específicas para múltiples sistemas embebidos. El apoyo ofrecido por los entornos actuales de desarrollo para adaptar la interfaz de usuario es limitado, así como la creación de interfaz de usuario en aplicaciones eficientes basadas en servicios Web. K. Klemisch [6] presentaron un trabajo sobre la adaptación de interfaz de usuario a diversos contextos. También presentó un nuevo *framework* y una herramienta para la reutilización de componentes de interfaz de usuario en entornos de desarrollo. M. Nabuco et al. [7] describieron un enfoque dinámico de ingeniería inversa para extraer los patrones de interfaz de usuario de aplicaciones Web existentes y facilitar la construcción de modelos de prueba en el contexto de PBGT (*Pattern Based GUI Testing*, Patrones de pruebas basadas en GUI).

La interfaz de usuario de un teléfono inteligente es diferente al de una tableta. Por esta razón, S. Chin et al. [8] mencionaron dos consideraciones: la Personalización basada en Estados y la Personalización basada en Proyectos. Se logró personalizar la interfaz de usuario de un teléfono inteligente a una tableta utilizando las consideraciones antes mencionadas. J. Engel et al. [3] descubrieron un método basado en patrones para transformar las interfaces de usuario de los sistemas interactivos a diversos contextos de uso. El objetivo principal del trabajo se encuentra en el modelado de las relaciones necesarias entre los diferentes patrones y la definición de un conjunto de patrones de transformación.

2.2. Uso de patrones de diseño de interfaz de usuario en diversos dominios y tipos de aplicaciones

A continuación se mencionan algunos trabajos relacionados con el uso de los patrones de diseño de interfaz de usuario en diversos dominios y tipos de aplicaciones como lo son la Web, los móviles y la televisión inteligente.

W. Jackson [9] informó sobre la creación de prototipos de interfaz de usuario, utilizando el software Pencil 2.0.5, y mostró cómo utilizar las funciones primarias del software para diseñar una interfaz de usuario para el sistema operativo Android. Además W. Jackson [10] mencionó la gran cantidad de diferentes tamaños de pantalla del dispositivo Android, densidades de píxeles, así como los posibles cambios de orientación del dispositivo. LinkedTV es un proyecto financiado por la Unión Europea que inició su trabajo en octubre de 2011, su objetivo es la "interconexión sin fisuras de la TV y la Web". L. Nixon [11] informó sobre la visión y el trabajo que realizó LinkedTV en su primer año.

El envejecimiento de la población es algo común en la sociedad, esto se vuelve relevante cuando se observa la importancia de la televisión en la vida diaria de las personas mayores. J. Coelho et al. [12] mencionaron las dificultades de los adultos mayores cuando se utilizan nuevas tecnologías y más específicamente, en lo relacionado con la visión y la audición. La televisión es una de las principales herramientas para el entretenimiento en el hogar y la información, pero en los últimos años la televisión cambió significativamente, así como su interacción entre los usuarios y el propio medio. A. Ingrosso et al. [13] presentaron las principales conclusiones de un test de usabilidad de una aplicación de una Televisión inteligente para T-commerce mediante la evaluación de los problemas de usabilidad de la aplicación.

En los trabajos de investigación revisados se enfatizó: 1) la importancia de ocupar patrones de diseño de interfaz de usuario y la manera en cómo facilitaron el desarrollo del software, 2) el poco uso de patrones de diseño de interfaz de usuario por parte de los desarrolladores debido a que no saben cómo ocuparlos o se usan incorrectamente al solucionar un problema.

3. Tipos de patrones de interfaz de usuario

3.1. Patrones de interfaz de usuario

Los patrones de diseño de interfaz de usuario se utilizan para facilitar el desarrollo de aplicaciones, porque las aplicaciones usan una serie de patrones que las hacen similares. Por lo tanto se pretende analizar diversos patrones de diseño, para diferentes dispositivos y plataformas, enfocando el uso de estos patrones al ámbito educativo y seleccionar aquellos que faciliten el desarrollo de una aplicación educativa y permitan que se visualice correctamente en distintos dispositivos. En la actualidad los sistemas operativos para dispositivos móviles más populares son Android, iOS, BlackBerry y Windows Phone. Cada sistema operativo tiene su propia identidad que se refleja en la apariencia y comportamiento de cada uno de los elementos gráficos. Sin embargo, todos comparten puntos de vista fundamentales que se manifiestan en el diseño de sus interfaces, como lo es la navegación, los cuadros de diálogo, notificaciones, entre otros. Se identificaron diferentes categorías de patrones de diseño de interfaz de usuario propuestos por diversos autores, para los dispositivos móviles se encuentran los planteados por Theresa Neil [14], Jennifer Tidwell [15], Mari Sheibley [16], Anders Toxboe [17] y UNITiD [18], en el caso de las televisiones inteligentes se encuentra LG Developer [19], Android TV [20] y Apple TV [21]. En la tabla 1 se presentan las categorías que cada autor asigna a sus patrones de diseño de interfaz de usuario.

Tabla 1. Categorías de patrones de diseño de interfaz de usuario presentado por autor.

Autor	Plataforma	Categorías	Patrones identificados
Theresa Neil	Móvil	Navegación (10); Formularios (7); Tablas y listas (8); Búsqueda, ordenamiento y filtración (14); Herramientas (7); Gráficos (8); Llamadas (8); Retroalimentación y ofrecimiento (5); Ayuda (3)	70
Jennifer Tidwell	Móvil y Web	Acciones del usuario (14); Organización del contenido (10); Navegación, indicadores y señalización (13); Organización de la página (13); Listas (12); Acciones y comandos (11); Árboles y gráficas (11); Formularios y controles (11); Medios sociales (12); Diseño móvil (11); Diseño visual (7)	125
Mari Sheibley	Móvil	Patrones de interfaz de usuario (22)	22
Anders Toxboe	Móvil y Web	Obtención de entrada (28); Navegación (25); Incorporación (9); Manejo de datos (11); Social (11); Diverso (5)	89
UNITiD	Móvil	Manejo de datos (16); Obtener entradas (8); Navegación (30); Notificaciones (7); Personalizar (6); Interacciones de pantalla (6); Social (4)	77
LG Developer	TV	Controles de interfaz de usuario (25)	25
Android TV	TV	Diálogo (1); Asistente (1); Configuraciones (1); Notificaciones (1)	4
Apple TV	TV	Barra de etiquetas (1); Tablas y Colecciones (3); Texto y búsqueda (3); Botones (2); Barra de navegación (1); Página de control (1); Indicadores de progreso (2); Segmentos de control (2); Alertas (2)	17

3.2. Análisis de clasificación de patrones de diseño de interfaz de usuario

A continuación en la tabla 2 se presenta el análisis para relacionar las plataformas, los patrones de diseño de interfaz de usuario que tienen en común y los autores que proponen los patrones de diseño.

Tabla 2. Relación entre patrones de diseño de interfaz de usuario

Plataforma	Patrones de diseño de interfaz de usuario	Autores
Móvil	Acordeón, Asistente, Barra de etiquetas, Barra de herramientas, Barra de progreso, Búsqueda, Carrusel, Dashboard, Diálogos, Formularios, Galería, Indicadores de carga, Listas, *Login*, Navegación por etiquetas, Notificaciones, Menú contextual, Segmentos de control, Tablas	Jennifer Tidwell, Mari Sheibley, Anders Toxboe y UNITiD
TV	Acordeón, Asistente, Barra de etiquetas, Barra de navegación, Barra de progreso, Botones, Búsqueda, Colecciones, Divisor, Formulario, Indicadores de carga, Listas, Notificación, Pagina de control, Selector expandible, Tablas, Vista detalle, Vista dividida	LG Developer, Android TV y Apple TV
Web	Acordeón, Asistente, Barra de progreso, Búsqueda, Carrusel, Dashboard, Diálogos, Formularios, Galería, Indicadores de carga, Listas, *Login*, Navegación por etiquetas, Notificaciones, Menú contextual, Segmentos de control, Tablas	Jennifer Tidwell y Anders Toxboe

Para desarrollar una aplicación se tiene que determinar el dispositivo y el tipo de aplicación que se desea desarrollar con el propósito de facilitar al usuario la interacción con la aplicación. J. Engel et al. [3] y K. Klemisch et al. [6] mencionaron en sus trabajos a los contextos de uso, donde un contexto de uso se relaciona con ciertos patrones de diseño de acuerdo a su funcionalidad. Por esta razón se realizó un análisis para reconocer patrones de diseño de interfaz de usuario en aplicaciones educativas de iOS y Android, para establecer contextos de uso orientados al ámbito educativo:

1. **Asistente:** Guía al usuario a través de un conjunto de reglas para configurar o realizar una tarea.
2. **Cuestionario:** Permite la generación de un cuestionario, encuesta, examen, entre otros.
3. **Curso:** Representa la estructura de un curso o materia, así como su descripción y contenido.
4. **Lista de contenido:** Hace referencia al conjunto de cursos o materias que contiene la aplicación.
5. ***Login*:** Utilizado para administrar la sesión de un usuario.

6. **Menú:** Ocupado para ver las opciones que se tienen sobre un elemento de la aplicación.
7. **Multimedia:** Se refiere a las imágenes, video y audios de la aplicación.
8. **Notificación:** Son aquellos mensajes de advertencia o error que emite la aplicación.

Tabla 3. Contextos de usos detectados al utilizar los patrones.

Contexto de uso	Plataformas	Patrones de diseño de interfaz de usuario relacionados
Asistente	Móvil, TV y Web	Asistente
Cuestionario	Móvil y Web	Formulario
	TV	Asistente, Barra de progreso
Curso	Móvil	Acordeón
	TV	Vista detalle
	Web	Acordeón
Lista de contenido	Móvil y Web	Lista, Acordeón
	TV	Listas, Colecciones
Login	Móvil, TV y Web	Iniciar sesión
Menú	Móvil	Menú contextual, menú emergente
	TV	Selector expandible, Barra de etiquetas
	Web	Menú contextual
Multimedia	Móvil	Carrusel, Galería, Rejilla de imágenes
	TV	Galería, Rejilla de imágenes
	Web	Carrusel
Notificación	Móvil y Web	Diálogo
	TV	Notificación

En la tabla 3 se muestra la relación entre los contextos de uso, las plataformas y los patrones de interfaz de usuario.

Los contextos de uso facilitan la selección de patrones de diseño de interfaz de usuario, permitiendo un uso adecuado y teniendo la confianza que diferentes aplicaciones educativas los utilizan en su interfaz gráfica.

4. Caso de Estudio: Desarrollo de una aplicación móvil de un curso de español utilizando los patrones de interfaz de usuario

A continuación se propone el diseño de la interfaz de usuario de una aplicación educativa para el aprendizaje electrónico utilizando un MOOC (Curso Online Masivo Abierto) que permita al docente impartir cursos de español para sus alumnos universitarios sobre los temas de lectura, redacción y ortografía. La aplicación se diseñó utilizando los contextos de uso propuestos para el ámbito educativo, la aplicación contempla lo siguiente:

1. **Aplicación para un dispositivo móvil:** La aplicación se ejecutará en un iPod o en un iPhone.
2. **Control de acceso:** Se necesita establecer un control de acceso para administrar el acceso y el progreso de los alumnos inscritos.
3. **Lista de cursos:** Se utiliza para mostrar los cursos que se impartirán en la aplicación.
4. **Temario del curso:** Hace referencia al contenido de temas y actividades que se realizarán en el curso.
5. **Conjunto de ejercicios por unidad:** Son aquellos ejercicios que se realizan en cada unidad.
6. **Examen por unidad y final:** Es el examen final de cada unidad del curso.
7. **Notificación de advertencias y logros:** En caso de haber alguna alerta o mensaje, se hará uso de notificaciones.
8. **Usabilidad:** Permitir que la aplicación sea fácil de usar, además de presentar de manera ordenada y sencilla el contenido.

La solución al caso de estudio se representa en la tabla 4, donde se hizo uso de los contextos de uso que se establecieron y se seleccionaron los patrones de interfaz de usuario para la plataforma de dispositivo móvil.

Tabla 4. Solución de caso de estudio.

Problema	Contexto de uso ocupado	Patrones de interfaz de usuario utilizados
Control de acceso	*Login*	Se utilizó el patrón de diseño iniciar sesión.
Lista de cursos	Lista de contenido	Se utilizó el patrón de carrusel y el patrón de lista.
Temario del curso y progreso del alumno	Lista de contenido	Se utilizaron los patrones de diseño barra de progreso, el patrón acordeón y el patrón de lista.
Ejercicios y exámenes	Cuestionario	Se utilizó el patrón de diseño de interfaz de usuario formulario en combinación con el patrón asistente.
Notificaciones	Notificación	Se utilizó el patrón diálogo, para el aviso de alertas y notificaciones.

En la figura 1 se observan los bosquejos de la aplicación educativa, utilizando los patrones de diseño de interfaz de usuario seleccionados por el contexto de uso.

Los patrones de diseño de interfaz de usuario son más entendibles al clasificarse en un contexto de uso, porque permite al diseñador escoger qué patrones de diseño son más adecuados para resolver el problema. Posteriormente a partir del bosquejo se procede a construir las interfaces gráficas de usuario, las cuales se visualizan en la figura 2.

Fig. 1. Bosquejos de aplicación educativa: a) Control de acceso; b) Lista de cursos disponibles; c) Temario del curso y progreso del alumno; d) Ejercicios, exámenes y progreso del alumno; e) Notificaciones.

El bosquejo facilitó el desarrollo de una aplicación educativa gracias a los beneficios que ofrecen los patrones de diseño de interfaz de usuario proporcionados por los contextos de uso. Además, es posible notar que un patrón de diseño proporciona la flexibilidad visualizar la apariencia que logrará alcanzar una aplicación utilizando un bosquejo, este a su vez se modifica con mayor facilidad que un prototipo funcional de la aplicación.

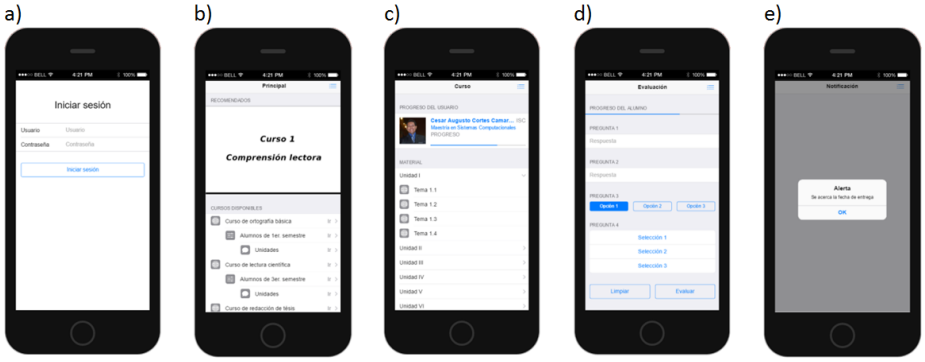


Fig. 2. Aplicación educativa: a) Control de acceso; b) Lista de cursos disponibles; c) Temario del curso y progreso del alumno; d) Ejercicios, exámenes y progreso del alumno; e) Notificaciones.

5. Conclusiones y trabajo futuro

Dado al avance tecnológico y la existencia de nuevos dispositivos, el diseñar una interfaz gráfica de usuario se ha vuelto una tarea más complicada, debido al gran número de dispositivos disponibles, esto genera un debate entre sus capacidades y sus limitaciones. Los contextos de uso orientados al ámbito educativo que se plantearon en este trabajo, facilita al diseñador de software escoger los patrones de diseño de interfaz

para una aplicación educativa de un celular inteligente, una tableta o una televisión inteligente.

También cabe resaltar que la televisión inteligente se ha vuelto muy popular hoy en día, y ahora se utiliza para diversos usos entre los cuales se encuentran el entretenimiento y el aprendizaje. Aunque todavía no tiene las mismas capacidades que un teléfono inteligente, tableta o PC se espera que en un futuro próximo tenga la misma experiencia de usuario, usabilidad y utilidad.

Como trabajo a futuro se pretende desarrollar una herramienta que haga uso de los contextos de uso y además se pretende investigar otros patrones de diseño y abarcar más dispositivos, con la finalidad de mejorar el número de patrones para impulsar el desarrollo de aplicaciones en el ámbito educativo.

Agradecimientos. Este trabajo fue patrocinado por el Consejo Nacional de Ciencia y Tecnología (CONACYT).

Referencias

1. Pressman, R. S.: Ingeniería del software un enfoque práctico. Séptima ed., McGraw-Hill (2010)
2. Alexander, C.: A pattern language. Estados Unidos de América, Oxford University Press (1977)
3. Engel, J., Märtin, C., Forbrig, P.: HCI Patterns as a Means to Transform Interactive User Interfaces to Diverse Contexts of Use. Human-Computer Interaction, Design and Development Approaches, USA, Springer Berlin Heidelberg, pp. 204−213 (2011)
4. Richard, J., Robert, J. M., Malo, S., Migneault, J.: Giving UI Developers the Power of UI Design Patterns. Human Interface and the Management of Information, Interacting with Information, Springer Berlin Heidelberg, pp. 40−47 (2011)
5. Kim, S.: Pattern and Event Based Logical UI Modeling for Multi-Device Embedded Applications. Convergence and Hybrid Information Technology, Springer Berlin Heidelberg, pp. 560−567 (2011)
6. Klemisch, K., Weber, I., Benatallah, B.: Context-Aware UI Component Reuse. Advanced Information Systems Engineering, Spain, Springer Berlin Heidelberg, pp. 68−83 (2013)
7. Nabuco, M., Paiva, A. C. R., Pascoal, J.: Inferring User Interface Patterns from Execution Traces of Web Applications. In: Computational Science and Its Applications – ICCSA 2014, Portugal, Springer International Publishing, pp. 311−326 (2014)
8. Chin, S., Iverson, D., Campesato, O., Trani, P.: Beyond Mobile: Tablets and TV. Pro Android Flash, Apress, pp. 399−426 (2011)
9. Jackson, W.: Android UI Design Concepts: Wire-framing & UI Layout Design Patterns. Pro Android UI, Apress, pp. 225−250 (2014)
10. Jackson, W.: Android UI Layout Conventions, Differences and Approaches. Pro Android UI, Apress, pp. 251−272 (2014)

11. Nixon, L.: Web and TV Seamlessly Interlinked: LinkedTV. Intelligent Technologies for Interactive Entertainment, Belgium, Springer International Publishing, pp. 32−42 (2013)
12. Coelho, J., Guerreiro, T., Duarte, C.: Designing TV Interaction for the Elderly – A Case Study of the Design for All Approach. A Multimodal End-2-End Approach to Accessible Computing, Springer London, pp. 49−69 (2013)
13. Ingrosso, A., Volpi, V., Opromolla, A., Sciarretta, E., Medaglia, C. M.: UX and Usability on Smart TV: A Case Study on a T-commerce Application. HCI in Business, Springer International Publishing, pp. 312–323 (2015)
14. Neil, T.: Mobile Design Pattern Gallery. Segunda ed., O'Reilly Media, pp. 284 (2012)
15. Tidwell, J.: Designing Interfaces. Segunda ed., O'Reilly Media, pp. 578 (2011)
16. Sheibley, M.: Mobile Patterns. Disponible: http://www.mobile-patterns.com (2013)
17. Toxboe: UI Patterns User Interface Design pattern Library. Disponible: http://ui-patterns.com (2016)
18. UNITiD.: Android Patterns. Disponible: http://unitid.nl/androidpatterns (2016)
19. LG ELECTRONICS: LG Developer. LG, Disponible: https://developer.lge.com/webOSTV/design (2013)
20. Android TV: Android TV Patterns. Disponible: http://www.google.com/design/spec-tv/patterns (2015)
21. Apple TV: Human Interface Guidelines. Disponible: https://developer.apple.com/tvos/human-interface-guidelines/ui-elements (2016)

ISSN 1870-4069

Despliegue de monitores con los mecanismos de reflexión y las extensiones de gestión de Java

Andoni Rodríguez-Díaz, Ulises Juárez-Martínez

Instituto Tecnológico de Orizaba, División de Estudios de Posgrado e Investigación, Orizaba, Veracruz, México

{arodriguezd,ujuarez}@ito-depi.edu.mx

Resumen. La monitorización permite a los sistemas auto-adaptables observar el estado actual de los mismos y de esta forma aplicar los cambios necesarios. El despliegue de monitores a un sistema implica añadir las instrucciones correspondientes en el código fuente y la posibilidad de afectación en el rendimiento durante la ejecución, sobre todo si los monitores emplean los mecanismos de reflexión. Este trabajo presenta un esquema para aplicar los monitores en los sistemas compilados, limitando el uso de reflexión.

Palabras clave: Monitores, reflexión, sistemas compilados.

Monitors Deployment Using Reflection Mechanisms and Java Management Extension

Abstract. Monitoring process let self-adapting systems to watch their current state and therefore making the needed changes. Deploying monitors into a system involves adding the necessary statements into the source code and the possibility to impact the system's performance during execution, also if monitors use reflection mechanisms. This paper features a scheme to deploy monitors on compiled systems, reducing the use of reflection.

Keywords: Monitors, reflection, compiled systems.

1. Introducción

Los sistemas auto-adaptables tienen la capacidad para modificarse a sí mismos en función del estado actual del entorno de ejecución, sin la necesidad de la intervención del usuario. IBM propuso el esquema de ciclo de control *MAPE-K*: monitorización, análisis, planeación y ejecución sobre una base de conocimiento; el cual es una secuencia de procesos para el diseño y desarrollo de sistemas auto-adaptables. Posteriormente, *MAPE-K* se integró en la arquitectura de "elementos autonómicos", el cual mediante el uso de sensores y actuadores se obtiene el estado de un sistema y se

aplican los cambios en los recursos de interés (elementos administrados), respectivamente [1,2].

La monitorización de un sistema permite obtener la información de su estado en tiempo de ejecución y en base a sus resultados aplican los cambios necesarios. Los sistemas distribuidos hacen uso de los monitores para la detección de errores imprevistos o para detectar modificaciones en el entorno de ejecución, ya sea desde la parte de software o hardware. Sin embargo, el despliegue de monitores a un sistema implica algunos inconvenientes como la modificación del código fuente o la generación de altos costos computacionales, lo que afecta en el rendimiento, sobre todo, si los monitores trabajan con mecanismos de análisis para obtener la información de la estructura del programa, como por ejemplo la reflexión. En [3] se enumeran algunas desventajas en cuanto al uso excesivo de la reflexión.

Este trabajo presenta un esquema que limita el uso de la reflexión para la monitorización y el despliegue de la misma en sistemas compilados. Este esquema se implementa con las herramientas de desarrollo de Java y el lenguaje AspectJ para la inyección de código en *bytecode*.

Este artículo se compone de la siguiente forma: la sección 2 se describen los trabajos relacionados con respecto a monitorización. En la sección 3 se describen brevemente las tecnologías que se utilizaron para la implementación de monitores. En la sección 4 se presenta el esquema de monitorización y la implementación del mismo. En la sección 5 se aplica la monitorización en un programa compilado. En la sección 6 se analizan los resultados. En la sección 7 se dan a conocer las conclusiones y el trabajo a futuro.

2. Trabajos relacionados

En [4] se presentó la herramienta *SPASS-meter*, la cual aplica la monitorización en los programas de las plataformas Java y Android, realiza análisis en tiempo de ejecución y presenta los resultados en tiempo real. En [5] se introdujo un marco de trabajo para la monitorización e instrumentación con capacidades de adaptación, dicho marco utiliza la interfaz de herramientas de la máquina virtual de Java (*JVM TI* por sus siglas en inglés) y los elementos de instrumentación de Java. En [6] se desarrolló una herramienta para construir modelos de rendimiento que se basan en el análisis en relación a las invocaciones en los comportamientos; esta herramienta es un agente que se aplica en la máquina virtual a través de la interfaz de *profiling* (*JVMPI*). En [2] se usó la reflexión y la característica de *proxy* dinámico para desplegar monitores, con la capacidad de adaptación en sí mismos, y realizar los trabajos de *logging* en cada invocación de un sistema. En [7] se presentó un enfoque que habilita la monitorización de programas en Java, con base en la compilación de *scripts* en código y su inclusión al sistema a través de AspectJ.

3. Tecnologías en Java

En esta sección se analizan las herramientas de programación que implementan la tecnología Java con la capacidad de interactuar con sistemas compilados.

3.1. Reflexión de Java

La característica de reflexión que ofrece Java permite realizar la introspección, que es la capacidad para analizar las estructuras de datos de un programa en tiempo de ejecución. Además, ofrece mecanismos para obtener información adicional tanto la descripción de una estructura de datos, así como también aquella en relación a los campos y métodos [8].

3.2. Extensión de gestión en Java

La extensión de gestión de Java (*JMX* por sus siglas en inglés) es una tecnología que ofrece dicha plataforma para el acceso a los recursos de los programas o servicios en ejecución. Los recursos se representan como objetos que se conocen como *MBeans*, los cuales se registran en un servidor que se denomina ``servidor *MBean*''; posteriormente estos recursos son accesibles de forma remota a través de los conectores que *JMX* ofrece, permitiendo la monitorización en los programas. Los objetos *MBean* se representan en forma estática durante la definición de una clase o de forma dinámica mediante la implementación de funcionalidades genéricas para el acceso a los datos de la clase de interés [9].

3.3. AspectJ

AspectJ es un lenguaje de programación orientado a aspectos, el cual extiende al lenguaje de programación de Java. A través del modelo de puntos de unión, AspectJ permite la separación de asuntos que afectan a distintas clases del programa principal y la encapsulación de los mismos en módulos, permitiendo la reutilización y facilitando el mantenimiento de software [10].

4. Esquema de monitorización propuesto

En esta sección se presenta el esquema para la monitorización de objetos en un sistema en tiempo de ejecución. El esquema implementa los elementos de la extensión de gestión de Java y los mecanismos de reflexión de la misma plataforma. En la figura 1 se muestra el diagrama de clases del modelo de monitorización. La clase Reflected obtiene la información de los campos y los métodos de la instancia de tipo java.lang.Class, por ejemplo java.lang.Object.

Una instancia de tipo Guardian interviene en el acceso a un objeto del sistema, dicho tipo implementa la interfaz DynamicMBean lo que permite su registro en el servidor *MBean* para la monitorización; además hace uso de la información de la clase Reflected para identificar a qué campo corresponde un valor.

El aspecto Adapting (Código 1, figura 2) realiza un corte en la llamada al constructor de una clase del sistema (línea 2), en el aviso around se construye el objeto (línea 7) y se pasa como parámetro para el constructor de la clase Guardian (línea 8), finalmente se devuelve la referencia del objeto que se construyó (línea 13).

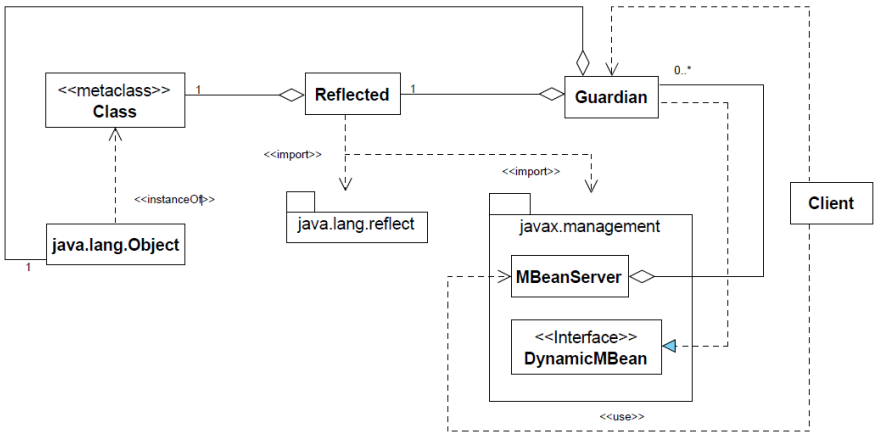


Fig. 1. Diagrama de clases del esquema de monitorización propuesto.

```
1  public aspect Adapting {
2      pointcut toBeAdaptable() : call([nombredelaclase].new(..));
3
4      Object around() : toBeAdaptable() {
5          Object object = null;
6          try {
7              object = proceed();
8              new Guardian(object);
9          }
10         catch (Exception ex) {
11             ex.printStackTrace();
12         }
13         return object;
14     }
15 }
```

Fig. 2. Aspecto para la construcción de un objeto de monitorización.

La clase Guardian (Código 2, figura 3) se compone de las referencias al objeto del sistema (línea 2) y al objeto Reflected (línea 3); durante el proceso de construcción, se obtiene la referencia a la instancia de tipo Reflected (línea 7), finalmente se agrega al servidor *MBean* (línea 8).

```
1  public class Guardian implements DynamicMBean {
2      private Object target;
3      private Reflected reflected;
4
5      public Guardian(Object target) {
6          this.target = target;
7          reflected = Reflected.getReflected(this.target.getClass());
8          ManagementServer.getServer().registerMBean(this);
9      }
10
11     //...
12 }
```

Fig. 3. Fragmento de código de la clase Guardian.

La clase Reflected (Código 3, figura 4) se compone de las referencias a la información de una clase perteneciente al sistema (líneas 3-5) y una referencia de tipo MBeanInfo, cuya información es de interés para el servidor *MBean*. La clase Reflected cuenta con un campo estático que representa el conjunto instancias de este tipo (línea 2), que se acceden a través de la invocación del método Reflected.getReflected() en la clase Guardian (Código 2); si una instancia de este conjunto coincide con la clase del objeto del sistema se devuelve la referencia, en caso contrario se crea la instancia y se aplica la reflexión para obtener la información del tipo del objeto y se devuelve la nueva referencia. Esto reduce el uso de la reflexión para consultar algún elemento de la estructura de una clase, en su lugar, dichas consultas se realizarán en los objetos que representan los campos y los métodos (línea 4 y 5).

```
1  public class Reflected {
2      private static Set<Reflected> reflectedSet = new HashSet<>();
3      private Class target;
4      private Map<String, Field> fields;
5      private Map<String, List<Method>> methods;
6      private MBeanInfo info;
7
8      //...
9  }
```

Fig. 4. Fragmento de código de la clase Reflected.

5. Ejemplo de monitorización

El ejemplo de monitorización consiste en obtener y visualizar las propiedades de las instancias de un tipo que se especifique en el aspecto Adapting (Código 1). Para visualizar la información de interés es necesario de un segundo programa, el cliente, que implemente los elementos de acceso que ofrece la tecnología *JMX* para acceder a los elementos que están bajo monitorización, el servidor. En este caso, se requiere visualizar las propiedades de cada instancia del tipo Cubo que forma parte de un programa para proyectar figuras en tres dimensiones. En el corte en punto del aspecto Adapting se define la clase Cubo (Código 4, figura 5).

```
1  public aspect Adapting {
2      pointcut toBeAdaptable() : call(treD.Cubo.new(..));
```

Fig. 5. Especificación de la clase Cubo en el corte en punto del aspecto Adapting.

Para el programa cliente (Código 5, figura 6) se obtiene la instancia al servidor *MBean* (línea 1). De esta conexión se consultan las instancias que se identifiquen con un prefijo en una expresión regular. Por cada instancia resultante de la consulta (línea 2), se muestra en pantalla el identificador completo de la misma, el nombre y el tipo del atributo que se compone en la clase de la instancia y su valor correspondiente (líneas 3-10).

```
1  MBeanServerConnection connection = getMBeanServerConnection();
2  connection.queryMBeans(new ObjectName("adapted.not.invaded.app:*"), null).
       forEach(i -> {
3      System.out.println(i.getObjectName() + " attributes:");
4      MBeanInfo info = connection.getMBeanInfo(i.getObjectName());
5      for(MBeanAttributeInfo attribute : info.getAttributes()) {
6          System.out.print("\tName: " + attribute.getName());
7          System.out.print(", Type: " + attribute.getType());
8          try {
9              Object value = connection.getAttribute(i.getObjectName(),
                   attribute.getName());
10             System.out.println(", Value = " + value);
11         }
12         catch (Exception ex) {
13             System.out.println(", Value = Not_Supported");
14         }
15     }}); //...
```

Fig. 6. Código para visualizar las propiedades de las instancias bajo monitorización.

Es necesario configurar las propiedades en la máquina virtual para permitir al cliente el acceso a los elementos de monitorización; principalmente el puerto de escucha bajo la propiedad -Dcom.sun.management.jmxremote.port=[puerto].

Al ejecutar el programa, se inicia el servidor *MBean* y se registran los objetos de monitorización. Posteriormente, el programa cliente accede al servidor y se muestra en pantalla la información que se requiere. En la figura 7 se muestra la salida resultante de la monitorización, se detectaron dos objetos del tipo Cubo (líneas 1 y 9), los cuales se identifican por el prefijo que se indicó anteriormente, el tipo de la instancia y el código *hash* de la misma; seguido de la lista de atributos de cada instancia.

```
1  adapted.not.invaded.app:type=treD.Cubo,at=64a294a6 attributes:
2      Name: lato, Type: float, Value = 2.0
3      Name: lung, Type: int, Value = 8
4      Name: poligoni, Type: [LtreD.Poligono;, Value = Not_Supported
5      Name: correnti, Type: [LtreD.Punto;, Value = [LtreD.Punto;@4ccabbaa
6      Name: origi, Type: [LtreD.Punto;, Value = Not_Supported
7      Name: posiz, Type: treD.Posicion, Value = Not_Supported
8
9  adapted.not.invaded.app:type=treD.Cubo,at=7e0b37bc attributes:
10     Name: lato, Type: float, Value = 3.0
11     Name: lung, Type: int, Value = 8
12     Name: poligoni, Type: [LtreD.Poligono;, Value = Not_Supported
13     Name: correnti, Type: [LtreD.Punto;, Value = Not_Supported
14     Name: origi, Type: [LtreD.Punto;, Value = Not_Supported
15     Name: posiz, Type: treD.Posicion, Value = Not_Supported
```

Fig. 7. Salida en pantalla del resultado de monitorización.

6. Resultados

El esquema de monitorización obtuvo la referencia de cada objeto del sistema compilado y del tipo de dato que se indicó en el aspecto (Código 1, figura 2), dicha referencia se encapsuló en una instancia y se registró en el servidor *MBean*. Durante el proceso, se analizó con la reflexión la estructura de datos del objeto perteneciente al sistema compilado, generando una referencia del análisis resultante, que posteriormente

se asoció a los objetos del mismo tipo, evitando aplicar nuevamente la reflexión, por lo tanto, se reduce el impacto al desempeño del sistema.

Se implementó un programa para acceder a los elementos de monitorización. La salida resultante de la figura 7 muestra el nombre y el tipo de cada propiedad, sin embargo, solo se visualizan los valores de las propiedades cuyos tipos son primitivos (líneas 2 y 3); al obtener los valores de las propiedades de tipos complejos, se lanza una excepción la cual indica que dicho tipo no es serializable, por lo tanto, ese valor no es accesible desde el cliente con los mecanismos de *JMX*. Una solución es incluir en la especificación de clases en el aspecto Adapting (Código 1, figura 2) aquellos tipos cuyos valores no son accesibles directamente por *JMX* y en su lugar aplicar la reflexión en tiempo de construcción.

7. Conclusión y trabajo a futuro

En este trabajo se presentó un esquema de monitorización para los sistemas compilados, el cual incorpora la reflexión de Java y los mecanismos de la tecnología *JMX*. Cuando se construye el primer objeto que corresponde a una clase, su estructura se analiza y se representa como una referencia, la cual se asocia en la creación de los siguientes objetos del mismo tipo, sin recurrir a la reflexión nuevamente. Esto reduce el impacto al desempeño del sistema evitando el uso de la reflexión para obtener la estructura interna de cada objeto.

Además, los elementos del sistema compilado se representan como instancias que *JMX* tomará en cuenta para los efectos de monitorización, evitando la modificación del programa en su representación en código intermedio. Como trabajo a futuro se implementarán mecanismos para simplificar las consultas de objetos en el servidor *MBean* y se analizarán alternativas para definir clases como puntos de unión en el aspecto para el despliegue de monitores.

Agradecimientos. Este trabajo cuenta con apoyo por parte del Consejo Nacional de Ciencia y Tecnología (CONACYT).

Referencias

1. Arcaini, P., Riccobene, E., Scandurra, P.: Modeling and analyzing mape-k feedback loops for self-adaptation. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS '15, Piscataway, NJ, USA, IEEE Press, pp. 13–23 (2015)
2. Dawson, D., Desmarais, R., Kienle, H. M., Müller, H. A.: Monitoring in adaptive systems using reflection. In: Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems, SEAMS '08, New York, NY, USA, ACM, pp. 81–88 (2008)
3. Eichelberger, H., Schmid, K.: Flexible resource monitoring of java programs. Journal of Systems and Software, pp. 163–186 (2014)

4. Wert, A., Schulz, H., Heger, C.: Aim: Adaptable instrumentation and monitoring for automated software performance analysis. In: Proceedings of the 10th International Workshop on Automation of Software Test, AST '15, Piscataway, NJ, USA, IEEE Press, pp. 38–42 (2015)
5. Harkema, M.: JPMT: A Java Performance Monitoring Tool. CTIT technical report series. Centre for Telematics and Information Technology, University of Twente (2003)
6. Colombo, C., Pace, G. J., Schneider, G.: Larva - safer monitoring of real-time java programs (tool paper). In: 2009 Seventh IEEE International Conference on Software Engineering and Formal Methods, pp. 33–37 (2009)
7. Forman, I. R., Forman, N.: Java Reflection in Action (In Action Series). Manning Publications Co., Greenwich, CT, USA (2004)
8. Oracle: Lesson: Overview of the jmx technology
9. Laddad, R.: AspectJ in Action: Practical Aspect-Oriented Programming. Manning Publications Co., Greenwich, CT, USA (2003)

ISSN 1870-4069

Análisis de las propiedades de corte aplicables sobre objetos funcionales

Jesús Juárez-De Felipe, Ulises Juárez-Martínez, María Antonieta Abud-Figueroa, José Luis Sánchez-Cervantes, Lizbeth Rodríguez-Mazahua

Instituto Tecnológico de Orizaba, División de Estudios de Posgrado e Investigación, Orizaba, Veracruz, México

jjuarezdefelipe@acm.org, ujuarez@ito-depi.edu.mx, mabud@ito-depi.edu.mx, jsanchezc@ito.depi.edu.mx, lrodriguez@itorizaba.edu.mx

Resumen. En lenguajes de programación como Scala y Java el modelo de programación orientado a objetos se ve fortalecido con el paradigma de programación funcional, así surge un nuevo paradigma basado en los objetos funcionales en donde cada función es un objeto de primera clase. Por lo que aparece la interrogante acerca de cómo programar aspectos en este tipo de paradigma, así como también de cómo aplicar patrones de diseño orientados a aspectos y cuáles son las ventajas y desventajas que se obtienen del empleo de la orientación a aspectos con objetos funcionales. La aportacion principal de este trabajo es describir cómo aplicar la programación funcional en los lenguajes Java 8 y Scala, también cómo aplicar la programación orientada a aspectos sobre los objetos funcionales y mediante algunos ejemplos se muestra hasta donde es posible usar el lenguaje AspectJ sobre los objetos funcionales.

Palabras clave: AspectJ, Java, Scala, objetos funcionales, programación orientada a aspectos.

Analysis of Cross-cutting Properties Applicable on Functional Objects

Abstract. In programming languages such as Scala and Java the object-oriented programming model is strengthened with the functional programming paradigm, so a new paradigm emerges based on the functional objects where each function is a first class object. So the question arises about how to program aspects in this type of paradigm, as well as how to apply aspect oriented design patterns and what are the advantages and disadvantages which are obtained from the use aspects orientation with functional objects. The main contribution of this work is to describe how to apply functional programming in Java 8 and Scala languages, how to apply the aspect oriented programming within functional objects and by some examples it is show to where it is possible to use the AspectJ language within functional objects.

Keywords: AspectJ, aspect oriented programming, functional objects, Java.

1. Introducción

El paradigma de programación objeto funcional surge de la combinación del paradigma de programación orientado a objetos junto con el paradigma de programación funcional, por lo que en este nuevo paradigma cada función se considera un objeto. Con la aparición de los objetos funcionales surge la interrogante sobre cómo encapsular adecuadamente los requerimientos no funcionales. Propiedades funcionales como las funciones de orden superior permiten igualar algunos conceptos de la programación orientada a objetos, por lo que se hace necesario revisar cómo se programan aspectos en este tipo de lenguajes híbridos, cómo se aplican los patrones de diseño [1, 2] orientados a aspectos (por ejemplo el patrón Objeto Trabajador [3] que convierte aplicaciones secuenciales en aplicaciones concurrentes) y cómo se obtienen ventajas (o desventajas) especialmente de un lenguaje como AspectJ que fue originalmente diseñado para Java.

Actualmente existen trabajos en los cuales se estudia la orientación a aspectos junto con lenguajes que se basan en el paradigma objeto funcional como lo son Scala y Java, por ejemplo en [4] se planteó la creación de ScalaPipe, que es un generador de aplicaciones de *streaming* para plataformas heterogéneas, mediante el uso de una colección de lenguajes de dominio específico incrustados en el lenguaje de programación Scala. En [5] se describió un marco de trabajo de programación orientada a aspectos totalmente funcional en el lenguaje Scala. En [6] se presentó el desarrollo de una biblioteca orientada a aspectos codificada en AspectJ, que pretende imitar el estándar OpenMP para la programación multi-núcleo en Java. Sin embargo, aún es necesario integrar adecuadamente todo este soporte para desarrollar soluciones eficientes para la industria.

La contribución principal de este trabajo consiste en mostrar cómo aplicar la programación orientada a aspectos usando el lenguaje AspectJ sobre los objetos funcionales en los lenguajes Java 8 y Scala, usando la sintaxis propia de AspectJ para Java y el sistema de anotaciones de AspectJ para Scala. Este artículo está organizado de la siguiente forma: en la Sección 2 se describe la aplicación del enfoque funcional desde un enfoque general. La Sección 3 muestra los objetos funcionales en el lenguaje Java 8 y cómo aplicar la programación orientada a aspectos. La Sección 4 muestra la implementación de la programación orientada a aspectos a los objetos funcionales en el lenguaje Scala. La Sección 5 muestra los resultados obtenidos. La Sección 6 da pie a la discusión de ideas. Finalmente en la Sección 7 se presentan las conclusiones y el trabajo a futuro.

2. Aplicación del paradigma funcional

El paradigma de programación funcional presenta varias ventajas [7], la más importante de ellas es que las funciones no presentan efecto de borde, esto quiere decir que no se modifica el estado interno de la función, no tiene variables globales o estáticas que sean modificadas y no presentan datos en la pantalla o realizan operaciones sobre archivos. Así al ser llamadas las funciones aún de manera concurrente estas no guardan ningún estado y los resultados que devuelvan las funciones serán siempre los mismos.

Esta característica de las funciones hace que el proceso de depuración de los programas sea mucho más rápido, ya que los programas que trabajan mediante funciones son la mayoría de las veces más confiables, además de que se presenta una mayor facilidad para la ejecución de los programas de manera concurrente y distribuida. Como desventaja del paradigma funcional se observó que en ocasiones es necesario que los métodos guarden alguna variable, por lo que en estos casos no conviene el uso de funciones, ya que se perderían las ventajas que el paradigma funcional aporta al aparecer el efecto de borde. Otra desventaja en el caso del lenguaje Java con el uso de objetos funcionales es la necesidad de utilizar la interfaz funcional para la declaración de los métodos y no declarar el comportamiento de manera directa.

Actualmente dos de los lenguajes orientados a objetos que han implementado el paradigma de programación funcional son Java y Scala. Comparando estos lenguajes de programación se observó que ambos cuentan con mecanismos para el control de concurrencia como son hilos, futuros y actores usando el *toolkit* Akka [13], aunque los últimos dos puntos Scala también los maneja de manera nativa. Además de que ambos lenguajes cuentan también con el manejo de funciones anónimas, funciones de orden superior y funciones de primera clase.

La ventaja que presenta Scala es que cuenta con un soporte para los mecanismos concurrentes antes mencionados más maduro, además de contar con soporte nativo para elementos importantes como los actores. Java tiene la ventaja de que AspectJ fue diseñado para él, por lo que su soporte está completo y se tienen todas las ventajas tanto del corte estático como del corte dinámico. Todas estas ventajas no se obtienen al trabajar con AspectJ mediante anotaciones.

3. Java 8

En Java los objetos funcionales se implementaron a partir de la versión 8 del lenguaje mediante expresiones lambda, y una forma de utilizarlas es mediante una interfaz funcional que como requisito sólo debe contener un único método abstracto. En el ejemplo que se muestra a continuación (Código 1) se emplea una lambda para realizar la operación de suma e imprimir el resultado, en la línea número 1 se observa una interfaz funcional llamada suma, que posee sólo un único método (línea 2), y en el método *main* de la clase se crea el objeto funcional, se le implementa su comportamiento (líneas 6 a 11) y por último se hace uso de él en la línea 12.

Para aplicar la programación orientada a aspectos sobre el lenguaje Java se utilizó el lenguaje AspectJ y para la compilación y ejecución de los códigos se utilizó el *plugin* de AspectJ para Eclipse [8].

```
1  interface Suma{
2      public int m(int x, int y);
3  }
4  public class Calculadora {
5      public static void main(String[] args) {
6          Suma a = new Suma() {
7              public int m(int x, int y) {
```

```
8               return x + y;
9               }
11        };
12        System.out.println(a.m(20, 10));
13    }
14  }
```

Código 1. Creación y uso de un objeto funcional en Java.

Para ver todos los puntos de unión con los que cada clase cuenta, Java proporciona la herramienta *javap*, que permite ver todo el *bytecode* que contiene cada archivo con extensión *.class* y así saber todos los posibles puntos de unión en dónde aplicar cortes para colocar avisos. Otra manera de ver todos los puntos de unión es directamente aplicando un aspecto sobre la clase interesada y usar la primitiva de AspectJ *whitin* [9] sobre toda la clase e imprimir todos los puntos de unión que encuentre. Los puntos de unión que AspectJ es capaz de detectar sobre el ejemplo anterior (Código 1) para la clase calculadora y que pertenecen al objeto funcional son:

1. initialization(paq.Suma())
2. call(int paq.Suma.sumar(int, int))
3. execution(int paq.Main.1.sumar(int, int))

En el ejemplo del codigo 2 se muestra cómo realizar cortes a todos los puntos de unión arriba mencionados.

El corte 1 se aplicó sobre el constructor de la interfaz suma (código 1 línea 6) y sólo muestra un mensaje, el corte 2 se realizó sobre la llamada al método sumar de la interfaz suma y permite capturar los valores que se le mandan al método y modificarlos. Para el corte 3 se tuvo que usar el comodín * dado que a la clase se le agregó por defecto un número 1, pero en la sintaxis de AspectJ para las firmas de los métodos los números no son válidos, este aviso captura el valor de retorno de la función. Analizando el bytecode con la herramienta javap se observó que el comportamiento del objeto funcional queda dentro de una clase interna en la clase calculadora (Código 1), pero no es posible acceder a ella mediante un aspecto, por lo que AspectJ no tiene el soporte adecuado para este tipo de construcciones.

```
1   public aspect CalculadoraAspect {
2       pointcut corte1():
3           initialization(paq.Suma.new(..));
4       pointcut corte2(int x, int y):
5           call(* paq.Suma.sumar(int, int)) && args(x, y);
6       pointcut corte3():
7           execution(int paq.Main.*.sumar(int, int));
8       before () :corte1(){
9           System.out.println("Suma.new");
10        }
11      int around(int x, int y): corte2(x, y) {
```

```
12            return proceed(x+1, y+2);
13        }
14          after() returning(Object r) :corte3(){
15            System.out.println("Retorno: "+r.toString());
16          }
17    }
```

Código 2. Aplicación de cortes al objeto funcional del Código 1.

De igual manera para la interfaz suma AspectJ no es capaz de detectar ningún punto de unión, aunque esta clase no es de gran importancia, ya que sólo contiene un método abstracto. Hay dos maneras de implementar las lambdas de forma más reducida. Con un estilo funcional:

```
Suma b = (x, y) -> x + y;
```

Y con un estilo más orientado a objetos:

```
Sum c= (x,y) -> {return x+y};
```

Los puntos de unión que AspectJ detecta para esta función son los siguientes:

1. call(int paq.Suma.sumar(int, int)),
2. execution(int paq.Main.lambda$0(int, int)).

El primer punto de unión es similar al primer ejemplo manejado (Código 1), pero el segundo es diferente, hace referencia a la ejecución de un método en una clase interna, está contiene el comportamiento de la función pero AspectJ no es capaz de detectarla.

Un problema que se presenta al trabajar expresiones lambda es cuando se vuelve a implementar el comportamiento de la misma interfaz funcional.

```
Suma a = (x, y) -> x + y;
Suma b = (x, y) -> x * y;
```

Los puntos de unión de la clase en donde se encuentran los dos objetos de la interfaz suma son:

1. call(int paq.Suma.sumar(int, int))
2. execution(int paq.Main.lambda$0(int, int))
3. call(int paq.Suma.sumar(int, int))
4. execution(int paq.Main.lambda$1(int, int))

Por lo que una manera de solucionarlo es validar la lambda que aparece primero en el código y esa es la numero cero, la siguiente es la numero 1 y así sucesivamente, para después verificar en el aspecto que el número de lambda sea el que se quiere cortar.

En el corte 5 se atrapa al objeto con la primitiva *target* para ver su clase, convertirla en cadena y validar que sea la lambda que se busca, en el corte 6 se valida directamente en el corte al metodo de la lambda en la cual se tiene interés.

```
1  public aspect MainAspect {
2      pointcut corte5(Object a):
3          call(int paq.Suma.sumar(int, int)) && target(a);
4      before(Object a): corte5(a) {
5          if(a.getClass().toString().contains("Lambda$1"))
6              System.out.println("Lambda 0");;
7          }
8        }
9      pointcut corte6():
10         execution(int paq.Main.lambda*$0(int, int));
11     before(): corte6() {
12         System.out.println("lambda 0 ");
13       }
14   }
```

Código 3. Aplicación de cortes para identificar una expresión lambda en particular.

4. Scala

Los objetos funcionales en Scala se trabajan mediante diferentes formas, ya que en Scala todo es un objeto [10] sólo se debe de cumplir el requisito de que ese objeto no presente efecto de borde. Scala cuenta con los *traits Function1* al *trait Function22* para definir funciones desde un argumento hasta veintidós, todas las funciones heredan de estos *traits* ya sea que se utilice la palabra reservada *extends* seguido del *trait* que corresponda a su número de párametros junto con sus valores de entrada y de retorno, o si no se indica la herencia de manera explícita a algún *trait*, Scala lo hace de manera autómatica.

El siguiente ejemplo muestra la aplicación de los objetos funcionales, en donde se emplea un objeto *singleton* que será el objeto funcional con su método *apply*, que es el método que de manera predeterminada utilizará la función para implementar su comportamiento.

```
1  object Test{
2      def main(args: Array[String]){
3          println(Suma(20, 30));
4      }
5  }
6 object Suma extends Function2[Int, Int, Int] { 7 def
apply(x: Int, y:Int): Int = x + y
8  }
```

Código 4. Objeto funcional en el lenguaje Scala.

Para la ejecución de los ejemplos se utilizó para la compilación y ejecución la herramienta de construcción SBT [11]. Usando las anotaciones de AspectJ [12] se

aplica la programación orientada a aspectos a programas escritos en el lenguaje Scala.

Al ser Scala un lenguaje que es interpretado por la máquina virtual de Java, todos los archivos con extensión *.scala* al compilarse dan como resultado uno o más archivos con extensión *.class* por lo que también se obtienen los puntos de unión que las clases contienen con los mismos procedimientos que se mostraron en la sección anterior Java 8. Al compilar el Código 4 se crean cuatro archivos con extensión *.class*, las clases que llevan en su nombre el símbolo "$" son las que toma la máquina virtual de Java para ejecutarlas y a estos archivos son a los que se deben de aplicar los aspectos.

Dado que el lenguaje Scala está hecho para reducir el código que los programadores escriben, siempre hay código que no se muestra a simple vista, por eso para aplicar los cortes es necesario revisar el *bytecode* de Java y ver a cuál instrucción corresponde cada uno de los puntos de unión que muestra AspectJ para conocer los nombres, escribir correctamente las firmas y realizar los cortes. Los puntos de unión que AspectJ es capaz de detectar para la función suma del Código 4 y su correspondiente nombre en *bytecode* son mostrados en la Tabla 1.

Tabla 1. Puntos de unión que AspectJ.

Punto de unión	Nombre en el bytecode
get(Suma. paq.Suma..MODULE$)	paq/Suma$.MODULE$:Lpaq/Suma$
call(int paq.Suma..apply$mcIII$sp(int, int))	paq.Suma$.apply$mcIII$sp:(II)I

Ahora una vez que se conocen los puntos de unión existentes en el código y cuál es su nombre completo en *bytecode* es posible generar las firmas colocando los correspondientes nombres de las clases y cambiando por comodines los símbolos que AspectJ no admite. Un ejemplo de un corte para los puntos de unión arriba mencionados se muestra a continuación:

```
1  @Aspect
2  class TestAspect {
3     @Before("get(paq.Suma$ paq.Suma$.MODULE*)")
4       def corte7(joinPoint: JoinPoint) = {
5           println("MODULE")
6       }
7     @Around("call(* paq.Suma$.apply$mcIII$sp(..)) && args(x, y)")
8       def corte8(joinPoint: ProceedingJoinPoint, x:Int,
        y:Int):Any = {
9         var a:Array[Object]=new Array[Object](2)
10        val z = new Integer(x+1)
11        val z2 = new Integer(y+2)
12        a(0)=z
13        a(1)=z2
```

```
14        joinPoint.proceed(a)
15
16      @AfterReturning(
17        pointcut = "call(* paq.Suma$.apply$mcIII$sp(..))",
18        returning= "result")
19          def corte9(joinPoint:JoinPoint, result:Object) {
20            println("Retorno : " + result);
21      }
22  }
```

Código 5. Aplicación de cortes al objeto funcional del Código 4.

El corte 7 es para el campo *module*, que es un objeto de tipo suma, el corte 8 sirve para cambiar el valor que recibe la función, se capturan los valores pero para mandarlos la función sólo recibe un arreglo de objetos por lo que es necesario crear el arreglo y llenarlo con los valores nuevos. El corte 9 captura el valor de retorno de la función y lo muestra. También AspectJ detecta puntos de unión en la clase *suma$* y estos están relacionados con la incialización de la clase suma, además de las llamadas y ejecución de los métodos que realizan las operaciones de la función. Otra manera de crear los objetos funcionales en el lenguaje Scala es mediante las funciones anónimas como se muestra a continuación.

```
1    object Test{
2        def main(args: Array[String]){
3  val Sum = (z:Int , y:Int) => z + y 4
println(Sum.apply(30, 40))
5            println(Sum(30, 40))
6        }
7    }
```

Código 6. Función anónima en el lenguaje Scala.

Al compilar el código se observa que se generan para la clase *test* del Código 6 tres archivos con extensión *.class*, dos de ellos tienen en su nombre el símbolo "$", uno contiene toda la implementación de la clase *test* del código y el otro archivo es una clase interna y contiene la implementación de la función anónima.

El problema que surge es con respecto a esta última clase, en donde AspectJ no es capaz de detectar ningún punto de unión, aunque al revisar el *bytecode* de esa clase se observa que contiene varios métodos y campos. Los puntos de unión que AspectJ encuentra para la clase test del Código 6 son los siguientes:

1. call(principal.Main.anonfun.1())
2. principal/Main$$anonfun$1.<init>:()V
3. call(int scala.Function2.apply$mcIII$sp(int, int))
4. scala/Function2.apply$mcIIsp:(II)I

En donde el primero es la llamada al constructor de la clase interna, este tipo de clases se generan automáticamente una por cada función anónima que se utilice. El segundo punto de unión es la llamada al método *apply*, a la firma de este método

internamente se le agregan los valores de entrada y de retorno, en este caso tres enteros (III).

5. Resultados

En las distintas pruebas que se realizaron mostradas en las secciones 3 y 4, se comprobó que AspectJ sí es capaz de realizar cortes sobre los objetos funcionales aunque con ciertas limitaciones. Existen algunas clases sobre las que AspectJ no cuenta con el soporte adecuado, por lo que aunque existan en el *bytecode* y sean ejecutadas por la máquina virtual son invisibles para los aspectos.

Estas clases que no son detectadas, son en su mayoría clases internas que se crean a tiempo de compilación y que no aparecen directamente en el código, el problema es que en varios de los casos éstas son las que contienen la implementación de los objetos funcionales, por lo que en esta situación sólo es posible aplicar cortes sobre las llamadas a los constructores y métodos de la clase externa pero no acceder a la clase interna.

En Java uno de los resultados que se obtuvo es que AspectJ no reconoce a las clases que se generan a tiempo de compilación y son las encargadas de manejar el comportamiento de las lambdas, éstas son clases generadas mediante la instrucción de *bytecode invoke dynamic*, esta instrucción sirve para la creación de clases en tiempo de compilación.

En Scala los resultados mostraron que este lenguaje al tener una sintaxis más corta para el programador, internamente genera una gran cantidad de código para crear todos los mecanismos que Scala necesita para funcionar, por lo que es necesario revisar el *bytecode* porque muchas de las líneas que en el código fuente son una, en el *bytecode* se generan varias y es necesario conocer exactamente en cuál se va a aplicar el corte, además de que internamente los nombres de algunos métodos presentan algunos cambios por lo que es necesario conocer su nombre real para crear la firma de ese método y realizar un corte.

6. Discusión

La programación orientada a aspectos presenta varias ventajas, entre ellas se encuentran el permitir una mayor reutilización del código, hacer que los programas sean más modulares y también más adaptables a los cambios, ya que no es necesario modificar el código fuente para cambiar el comportamiento de un programa, además permitir una mejor modularización del código. Todas estas ventajas que proporciona la programación orientada a aspectos se suman a las que proporcionan los objetos funcionales, por lo que es necesario un soporte adecuado de la programación orientada a aspectos en el lenguaje AspectJ para los objetos funcionales en los lenguajes Java y Scala.

7. Conclusiones

La programación con objetos funcionales tiene la ventaja de ofrecer una mejor modularización. Así como también al tener las funciones un estado inmutable pueden

ser llamadas de manera concurrente y al no tener las funciones variables adentro de ellas que pudieran alterarse cada vez que estas son llamadas, siempre devolverán los mismos resultados.

La programación orientada a aspectos con objetos funcionales no tiene todavía un gran soporte por parte del lenguaje AspectJ, pero con lo que se tiene hasta este momento es posible realizar varias tareas como interceptar los valores que se le envían a los objetos funcionales y cambiarlos o capturar los valores que las funciones retornan. Como trabajo futuro se aplicará la programación orientada a aspectos a las clases que son generadas automáticamente y que manejan el comportamiento de los objetos funcionales y que en este momento AspectJ no cuenta con el soporte adecuado, por lo tanto no es capaz de identificar puntos de unión dentro de ellas. Todo esto permitirá plantear herramientas y/o extensiones al lenguaje AspectJ para que sea posible aplicar correctamente aspectos a los objetos funcionales en los lenguajes Java 8 y Scala.

Agradecimientos. Los autores agradecen al Tecnológico Nacional de México por apoyar este trabajo. Además, este trabajo de investigación fue patrocinado por el Consejo Nacional de Ciencia y Tecnología (CONACYT), así como por la Secretaría de Educación Pública (SEP) a través de PRODEP.

Referencias

1. Hunt, J.: Scala Design Patterns: Patterns for Practical Reuse and Design. Springer Publishing Company, Incorporated (2013)
2. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns; Elements of Reusable Object-Oriented Software. Addison Wesley (1994)
3. Laddad, R.: AspectJ in Action Practical Aspect-Oriented Programming. Greenwich, CT: Manning Publications Co. (2003)
4. Wingbermuehle, J. G., Chamberlain, R. D., Cytron, R. K.: ScalaPipe: A Streaming Application Generator. Field-Programmable Custom Computing Machines (FCCM). In: IEEE 20th Annual International Symposium, pp. 244–244 (2012)
5. Spiewak, D., Zhao, T.: Method Proxy-Based AOP in Scala. Journal Of Object Technology, Vol. 8, No. 7 (2009)
6. Medeiros, B., Sobral, J. L.: Implementing an OpenMP-like standard with AspectJ. In: Proceedings of the 3rd workshop on Modularity in systems software (MISS '13), ACM, New York, NY, USA, pp. 1–6 (2013)
7. Wampler, D.: Functional Programming for Java Developers. O'Reilly (2011)
8. AspectJ Development Tools. Online, http://www.eclipse.org/ajdt (2016)
9. Miles, R.: AspectJ Cookbook. Sebastopol, CA, O'Reilly (2004)
10. Odersky, M., Spoon, L., Venners, B.: Programming in Scala. Artima (2007)
11. The interactive build tool. Online, http://www.scala-sbt.org (2016)

12. An Annotation Based Development Style. Online, https://eclipse.org/aspectj/doc/next/adk15notebook/ataspectj.html (2016)
13. Akka. Online, http://akka.io (2016)

ISSN 1870-4069

Inteligencia de negocios y minería de datos aplicado a la industria refresquera

Fani Rodríguez Flores[1 2], Leticia Flores Pulido [1], Emiliano Dávila de la Rosa[2]

[1]Universidad Autónoma de Tlaxcala,
Departamento de Ciencias Básicas, Ingeniería y Tecnología,
México

[2]Ajemex, Big Cola, México

Resumen. El presente articulo sintetiza el desarrollo de una solución de minería de datos para la inteligencia de negocios aplicado al canal detalle, en una industria refresquera. El objetivo es desarrollar el análisis, diseño y construcción de una solución siguiendo los lineamientos de la metodología CRISP-DM. La solución de inteligencia de negocio que se utilizó, cuenta con un data mart con información consolidada, lo cual permitirá a los usuarios acceder más rápido a la información perteneciente a los indicadores comerciales, de esta forma los usuarios podrán tomar decisiones que concuerden con la realidad del entorno comercial de la empresa.

Palabras clave: Minería de datos, CRISP-DM, indicadores comerciales.

Business Intelligence and Data Mining Applied to the Refreshment Industry

Abstract: This paper summarizes the development of a data mining solution for business intelligence applied to the channel detail in a soft drink industry. The aim is to develop the analysis, design and construction of a solution based on the guidelines on the CRISP-DM methodology. The solution of business intelligence used has a data mart with consolidate information, which allows users to quickly acces the corresponding commercial indicators information so the users can make decisions that match the reality of the business.

Keywords: Data mining, CRISP-DM, commercial indicators

1. Introducción

AJEMEX es una empresa refresquera, que pertenece al grupo de AJE es una empresa multinacional de bebidas, con presencia en más de 20 países de Asia, Latinoamérica y África, Cuenta con múltiples canales de distribución para llegar a distintos mercados o

segmentos. El Canal Detalle cuenta con una fuerza de ventas que se encarga de hacer contacto con los minoristas (detallistas), que venden los productos al público.

La empresa AJEMEX cuenta con diferentes Bases de Datos para poder almacenar su información, los datos de la preventa se almacenan en la base de datos de Aplicativo móvil mediante el dispositivo móvil, esta base de datos cuenta con más de 500,000 registros. La bases de datos donde se almacena la información procedente de los sistemas de Planeación de recursos empresariales (en inglés ERP, Enterprise Resource Planning) en la cual se realiza la facturación electrónica, esta base de datos cuenta con más de 3, 000,000 de registros.

Los usuarios necesitan de información segura y detallada, de cómo se realiza la atención al cliente en el canal detalle. Para realizar este análisis los usuarios solicitan al departamento de Tecnologías de la información la generación de diferentes reportes y tablas dinámicas, estas herramientas no proporcionan la facilidad de analizar la información, para poder ayudar a definir estrategia.

Los usuarios que toman decisiones en las ventas a detalle, se basan en los indicadores de mercado para poder definir estrategias de negocio para ciertos productos o fuerzas de ventas. La calidad, disponibilidad y presentación de la información juegan un papel decisivo en la toma de decisiones. Actualmente, se tiene un enfoque tradicional para el análisis de los datos donde se tienen fuentes de datos estructuradas y fuentes de datos no estructuradas.

Al aplicar técnicas de agrupamiento de minería de datos y análisis estadísticos para la obtención de información referente a los indicadores comerciales, es posible agilizar la definición veraz y oportuna de los indicadores para que sea más eficiente la toma de decisiones en el canal de venta a detalle, construyendo una Solución de Minería para Inteligencia de Negocios aplicado a canal detalle (SMINACD). El propósito es que la información relacionada con dichos indicadores pueda ser obtenida de manera eficiente veraz y oportuna. De esta forma, los usuarios podrán tomar decisiones que concuerden con la realidad del entorno comercial de la empresa.

2. Trabajos relacionados

2.1. Sistema de inteligencia de negocios para acueductos y alcantarillado

En el trabajo de Vanegas [1], los sistemas de procesamiento de transacciones (OLTP), dentro de los que se incluyen los sistemas de gestión, tienen diferentes limitaciones en cuanto a la forma de analizar la información y la usencia de reportes que muestren la información necesaria para la toma de decisiones. Mediante el uso de Inteligencia de mercados (BI) se logra unir el mundo de los datos y el de los negocios, esta permite a las empresas analizar grandes cantidades de datos de forma rápida y sencilla a la vez que facilita y apoya el proceso de toma de decisiones (PTD).

Se desarrolló e implemento un Sistema de Inteligencia de Negocios para las condiciones específicas de la EAALG (Empresa de Acueducto y Alcantarillado de la provincia Granma) permitirá mejorar la disponibilidad de información para el apoyo al PTD.

La metodología a utilizar es la HEFESTO de la cual se utilizaron las mejores prácticas, la metodología está compuesta por seis fases inicio, análisis de los

requerimientos, análisis de las fuentes de datos, modelado del DWH, integración de datos, representación de la información.

2.2. Minería de datos educativa

En el trabajo de Ballesteros [2], con el transcurso de los años, las actividades y el desarrollo de nuevas tecnologías se ha generado de forma considerable el almacenamiento de información, donde todo ese flujo de información que sea recolectado ha permitido satisfacer las necesidades diarias de las organizaciones, pero ha presentado un problema inherente en las capacidades humanas para analizar y transformar la información en conocimiento útil y relevante que apoye a la toma de decisiones, las técnicas y métodos de lo que se ha denominado actualmente como Minería de Datos (DM) enfocada a la gestión educativa, Minería de Datos Educativa (Por sus siglas en ingles EDM (Educational Data Mining), esta herramienta servirá a docentes que requieran evaluar sus prácticas y metodologías desarrolladas e implementadas con sus estudiantes, ya sea mediante el uso de agentes tutores inteligentes, sistemas virtuales de educación o estrategias activas de aprendizaje dentro del aula.

Se describe uno de los métodos analíticos para la selección de atributos y la extracción de parámetros de información KDD (Knowledge Data Discovery) conocido como el proceso de Descubrimiento de Conocimiento en Bases de Datos es un proceso no trivial de identificación de patrones válidos, novedosos y potencialmente útiles sobre un conjunto de datos, esto es, el objetivo es encontrar conocimiento útil, valido, relevante y nuevo sobre una determinada actividad, las etapas son Pre procesamiento, búsqueda (la utilización de técnicas y métodos de DM (Minería de datos)) y evaluación.

2.3. Un Modelo de Procesos para Proyectos de Explotación de Información

En el trabajo de Vanrell [3], dentro del desarrollo de proyectos de una empresa, los denominados proyectos de exploración de la información, poseen características propias que los hacen diferentes del resto, y no existe un modelo de procesos que se ajuste a este tipo de proyectos. COMPETISOFT es la proyección a nivel iberoamericano del modelo de procesos para el desarrollo de software MoProSoft, el modelo inicial fue modificado y adecuado a las necesidades de otros países, se le incorporo el modelo de evaluación EvalProSoft y se definieron niveles de madurez. La metodología CRISP-DM se encuentra definida en base a un modelo jerárquico de procesos.

El objetivo es crear un modelo de procesos de explotación de información orientado a Pymes tomando como base el modelo COMPETISOFT es la proyección a nivel iberoamericano del modelo de procesos para el desarrollo de software MoProSoft, esta metodología define un ciclo de vida de los proyectos de explotación de información que define las principales fases de un proyecto de este tipo. Estas fases son entendimiento de negocios, entendimiento de los datos, preparación de los datos, modelado, evaluación, despliegue.

3. CRISP-DM

La metodología de CRISP-DM para Minería de Datos [4], esta descrita en términos de un modelo de proceso jerárquico, consistente en un conjunto de tareas descritas en cuatro niveles de abstracción (de lo general a lo específico): fase, tarea genérica, tarea especializada, e instancia de procesos. La metodología CRISP-DM estructura el ciclo de vida de un proyecto de Minería de Datos en seis fases, que interactúan entre ellas de forma iterativa durante el desarrollo del proyecto como se muestra en la Fig. 1.

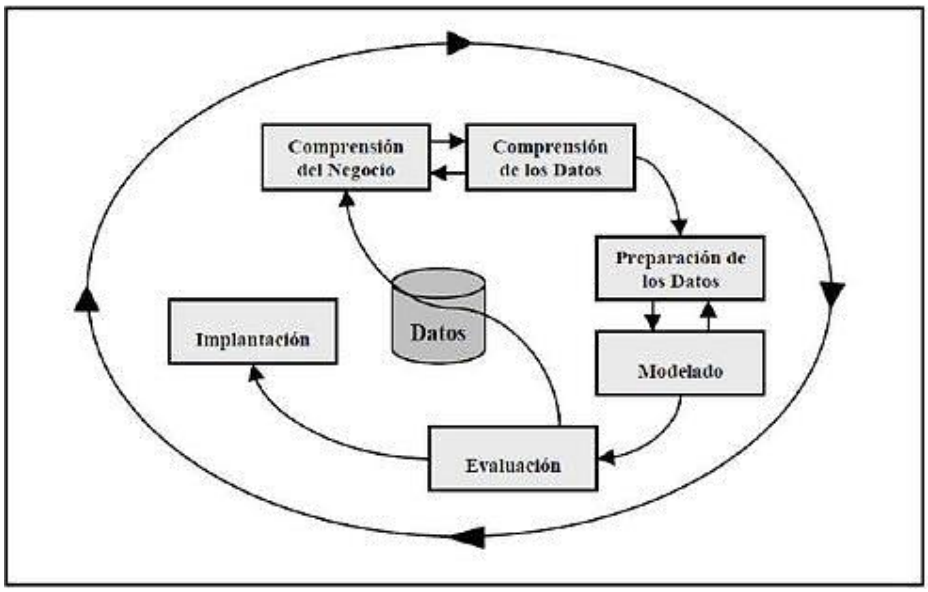


Fig. 1. Fases del proceso de modelado metodología CRISP-DM [4].

Comprensión del Negocio: La primera etapa de la metodología apunta a la comprensión de los objetivos del proyecto desde el punto de vista de los objetivos de negocio. En esta etapa se desarrollan los pasos preliminares para lograr los objetivos del negocio con herramientas de minería de datos.

Comprensión de Datos: En esta fase se desarrolla el entendimiento de datos y todas aquellas actividades relacionadas con la limpieza de datos, identificación de problemas vinculados con la toma de datos, procedimientos para determinar la calidad de datos y todo lo tendiente a facilitar la familiarización con los datos.

Preparación de los Datos: En esta etapa se desarrollan las actividades para construir el conjunto de datos final. Aquí se relaciona directamente el formato de los datos obtenidos con las herramientas de minería de datos a utilizar.

Modelado: En esta fase se eligen diferentes técnicas de modelado de datos y se estudian y ajustan parámetros con los valores correctos para el proyecto, Es altamente probable que desde el modelado sea necesario volver a la fase de preparación de datos puesto que todas las técnicas en evaluación pueden tener requisitos de formato de datos diferentes.

Evaluación: Hasta esta fase se han obtenido algunos modelos de minería de datos con sus datos y parámetros establecidos en forma óptima pero antes de pasar a la etapa final es necesario evaluar los resultados obtenidos por la ejecución de los programas en función de los objetivos de negocio. Aquí se puede presentar la necesidad de eliminar, modificar o considerar nuevas cuestiones relacionadas con el negocio.

Despliegue: El final del proyecto no termina con el modelado de datos y su ejecución y posterior evaluación de los resultados puesto que el conocimiento obtenido hasta aquí debe ser presentado de forma clara y precisa a todos los actores dentro de la organización, se puede presentar un simple informe de resultados, desarrollar una aplicación para la presentación de los resultados o bien instruir al usuario de los modelos para que ellos mismos generen y ejecuten los modelos con nuevos datos. Es importante al final de esta fase tener desarrollada toda la documentación del proyecto para dar independencia al usuario final en la utilización y generación de nuevos procesos de explotación de datos.

4. Minería de datos

La minería de datos puede definirse inicialmente como un proceso de descubrimiento de nuevas y significativas relaciones, patrones y tendencias al examinar grandes cantidades de datos [5]. La disponibilidad de grandes volúmenes de información y el uso generalizado de herramientas informáticas ha trasformado el análisis de los datos orientándolos hacia determinadas técnicas especializadas englobadas bajo el nombre de minería de datos o Data Minning.

Las técnicas de minería de datos persiguen el descubrimiento automático del conocimiento contenido en la información almacenada de modo ordenado en grandes bases de datos. Estas técnicas tienen como objetivo descubrir patrones, perfiles y tendencias a través del análisis de los datos utilizando tecnología de reconocimientos de patrones, redes neuronales, lógica difusa, algoritmos genéticos y otras técnicas avanzadas de análisis de datos [5].

5. Inteligencia de negocios

El contexto de la sociedad de la información ha propiciado la necesidad de tener mejores, rápidos y más eficientes métodos para extraer y trasformar los datos de una organización en información y distribuirla a lo largo de la cadena de valor [6]. La inteligencia de negocios (o Bisiness Intelligence) responde a dicha necesidad, y podemos entender, que es una evolución de los sistemas de soporte a las decisiones (DSS Decissions Suport Systems). La inteligencia de Negocios busca transformar los datos en información para finalmente transformar la información en conocimientos. Algunas de las tecnologías que forman parte de Inteligencia de Negocios son data werehouse, reporting, análisis OLAP(On-Line Analytical Processing), análisis visual, análisis predictivo, cuadro de mando, cuadro de mando integral, minería de datos, gestión del rendimiento.

6. Aplicación de la metodología propuesta

Para el desarrollo de la solución de minería e inteligencia de negocios aplicado al canal detalle, se utilizó la metodología CRISP.DM, fue adaptada a las necesidades del proyecto. Las etapas que comprende la metodología CRISP-DM son las siguientes:

6.1. Comprensión del negocio

La primera etapa de la metodología apunta a la comprensión de los objetivos del proyecto los cuales son:

- Disponer de una base de datos que permita extraer conocimiento de la información histórica almacenada en la organización.

El contenido del data warehouse deben ser entendibles, navegables y su acceso debe estar caracterizado por el alto rendimiento. En esta etapa se obtuvieron los requerimientos de la solución como se muestra en la Tabla 1.

Tabla 1. Requerimientos de la solucion.

Requerimiento	Necesidad
RF1	Obtener información en diferentes niveles de consulta
RF2	Obtención de indicadores comerciales: Cobertura, Efectividad de Visita, Drop Size, Liquidación, Efectividad de Compra.
RF3	Exportar información o cuadros de mando de los resultados.
RF4	Proveer una solución que permitirá a los usuarios finales generar sus propios reportes.
RF5	La solución deberá tener los mecanismos para controlar la seguridad de la Información.

7. Resultados de inteligencia de negocios

En la Empresa Ajemex se necesita conocer en qué zona y ruta se realiza mayor venta, para poder realizar la asignación de vendedores, correspondientes, se realiza en análisis de WEKA, contiene las herramientas necesarias para realizar transformaciones sobre los datos, tareas de clasificación, regresión, clustering, asociación y visualización [10].

Para el desarrollo de esta solución se analizaran los datos provienen de la tabla ventas a nivel ruta por mes, la cual contienen 718,324 registros ,cuenta con una serie de atributos los cuales son número de región, código de la estructura, indicador, año, mes, tota, factor 1, factor 2, cuota por mes. Para el análisis de los datos se utiliza la técnica de árboles de decisión en la construcción del modelo a partir de los datos.

Un árbol de decisión es una estructura que permite dividir un extenso conjunto de datos relacionados entre s en conjuntos más pequeños de datos mediante la aplicación secuencial de sencillas reglas de decisión [7]. Se construir un primer clasificado para los datos, el algoritmo J48-C 25-M 2 de WEKA, El cual es una implementación del algoritmo C4.5, uno de los algoritmos de minería de datos más utilizado.

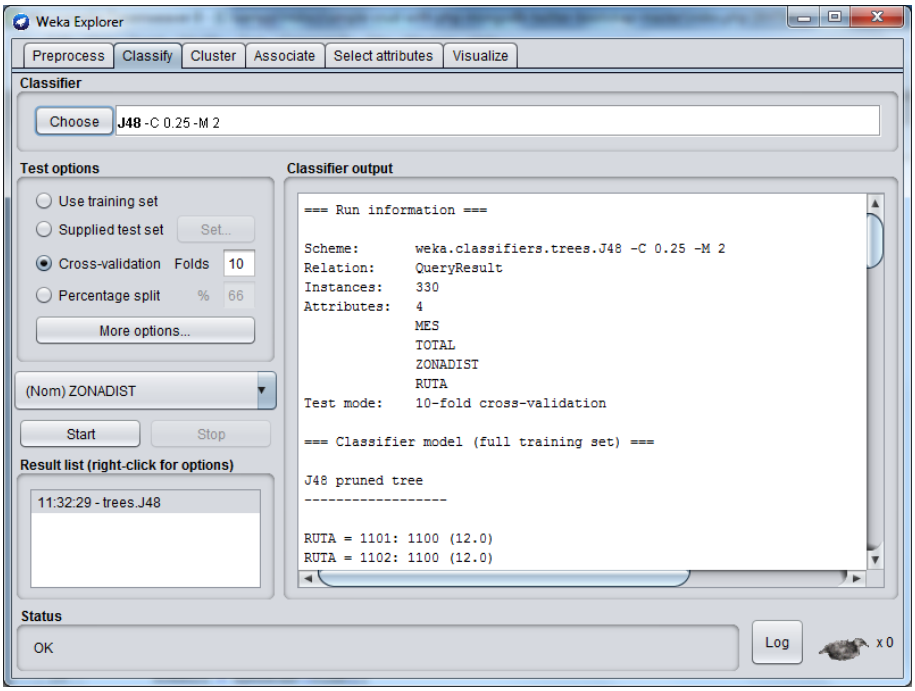


Fig. 2. Visualización para el análisis de datos por ZONADIST.

Para poder Realizar el análisis de los datos se realizaron pruebas por Zona de distribución, ruta, Mes. En la Fig.2, muestra información sobre el tipo de tipo de clasificador utilizado, el archivo el cual se está trabajando, el número de instancias (330), el número de atributos 4, y sus respectivos nombres.

En la Fig. 3. se muestra la matriz de confusión por Zona de Distribución (ZONADIST), la cual podemos observar que los valores de la diagonal son los aciertos y el resto lo errores, podemos observar que la zona 1100, se clasifican correctamente 89.

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
 89  0  0  0  0  0  0  0 |  a = 1100
  0 28  0  0  0  0  0  0 |  b = 1200
  0  0 35  0  0  0  0  0 |  c = 1300
  2  0  0 62  0  0  0  0 |  d = 1400
  0  0  0  0 24  0  0  0 |  e = 5000
  0  0  0  0  0 40  0  0 |  f = 1500
  0  0  0  0  0  0 30  0 |  g = 1600
  0  0  0  0  0  0  0 20 |  h = 1700
```

Fig. 3. Matriz de confusión por ZONADIST.

En la Tabla 2 se muestra una comparación del porcentaje de clasificación de las instancias, en la cual podemos observar que la clasificación por ZONADIST, nos proporciona un mejor porcentaje de instancias correctas.

Tabla 2. Tabla de hechos.

Nombre	Necesidad	Número de Registros
VTASRUTAM	Ventas a nivel ruta por mes	71,823
VTASRURAD	Ventas a nivel ruta por d a	127,4761
VTASCEDID	Ventas a nivel cedi por d a	44,471
VTASCEDIM	Ventas a nivel cedi por mes	1,885

7.1. Herramienta de inteligencia de negocio PowerPivot

PowerPivot es una verdadera revolución dentro del análisis de datos porque nos da todo el poder que se necesita para realizar complejos análisis sin requerir la intervención de técnicos. Esta herramienta, un complemento Excel, que contiene una base de datos tipo OLAP en una memoria interna [8].

Para que los usuarios finales puedan tomar decisiones que concuerden con la realidad de la empresa, utilizamos la herramienta PowerPivot la cual transforma una solución de inteligencia de negocio impulsada por usuarios, sin depender del departamento de TI, para la generación de tablas dinámicas, cuadros de mando, reportes, etc. Al seleccionar las tablas de hechos y dimensiones, enseguida se despliega los datos de las tablas seleccionadas. PowerPivot realiza una copia de los datos en Excel. Millones de datos se pueden almacenar de manera efectiva por PowerPivot utilizando una nueva tecnología que comprime los datos. Esta herramienta nos permite exportar a Excel, la información de diferentes maneras, en la Fig 4, se muestra un cuadro de mando, el cual nos muestra información de los indicadores comerciales, la información cambia en base al nombre de la sucursal, mes y año.

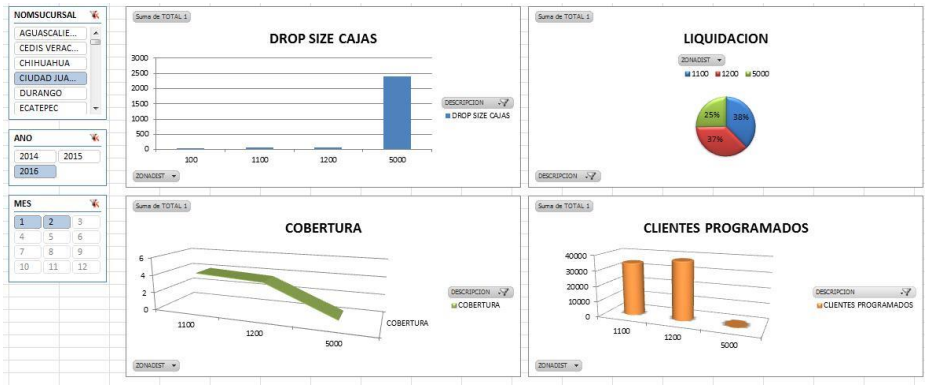


Fig. 4. Cuadro de mando con los indicadores comerciales.

8. Conclusiones y trabajos futuros

Hoy las empresas manejan gran cantidad de datos, se necesitan diferentes herramientas para poder convertir los datos en información, la cual el usuario pueda explotar y analizar. Para poder tomar decisiones.se desarrollo una solución para inteligencia de negocios aplicado a canal detalle, para la empresa refresquera Ajemex, se realizó un análisis de la información referente a los indicadores comerciales, mediante la metodología CRISP-DM. Al término del análisis se obtuvo como resultado un Data Marts de ventas, el cual obtiene la información de una manera más rápida. Se utilizaron técnicas de minería de datos, para poder clasificar la información, tales como arboles de decisión (algoritmo J48), se utilizaron los métodos para poder obtener información referente a las ventas por zona.

La Metodología CRISP-DM, facilito el poder realizar un proyecto de exploración de la información, al utilizar herramientas como PowerPivot se le proporciona al usuario la capacidad de poder realizar su análisis y reportes sin necesidad de depender del área de TI, esta herramienta es fácil de utilizar ya que los usuarios están familiarizados con Excel, solo será un complemento para poder realizar sus diferentes reportes. En trabajos futuros se puede utilizar el algoritmo analizado, para poder integrarlo al PowerPivot y nos pueda ayudar a predecir como será la venta en diferentes rutas.

Referencias

1. Vanegas, E.: Sistema De Inteligencia de Negocios Para Acueductos y Alcantarillado. 3C TIC, 7ma Edición (2013)
2. Ballesteros, A.: Minería de datos educativa: Una herramienta para la investigación de patrones de aprendizaje sobre un contexto educativo. Lat. Am. J. Phys. Educ., Vol.7
3. Vanrell, J. A.: Un Modelo de Procesos para Proyectos de Explotación de Información. Lacrest, Medellin (2012)
4. Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., Wirth, R.: CRISP-DM 1.0 step-by-step data mining guide. Thechnical report (2000)
5. Pérez, C.: Minería de Datos, Técnicas y Herramientas. Thomson Ediciones Paraninfo, S.A., 1ra Edición (2007)
6. Conesa, J.: Introducción al Business Intelligence. Editorial UOC, 1ra Edición (2010)
7. Han, J., Kamber, M.: Data mining: concepts and techniques. Morgan Kaufmann. United States of America (2016)
8. Muni, L.: PowerPivot con Excel a su alcance para convertir sus datos en información. Pro t Editorial I., S.L., Barcelona (2012)
9. Portada sobre la plataforma Pentaho Open Source Business Intelligence, En línea, http://pentaho.almacen-datos.com/kettle-spoon.html
10. Manual de WEKA. En línea, http://sci2s.ugr.es/sites/default/ les/ les/ Teaching/GraduatesCourses/InteligenciaDeNegocio/weka.pdf

ISSN 1870-4069

Generación automatizada de aplicaciones inmóticas bajo el enfoque de LPS

Jesús-Moisés Hernández-López, Ulises Juárez-Martínez

Instituto Tecnológico de Orizaba,
Veracruz, México

yeste_dracker@hotmail.com, ujuarez@ito-depi.edu.mx

Resumen. Una línea de productos de software (LPS) se refiere a los métodos de ingeniería de software, herramientas y técnicas que se utilizan para crear un conjunto de sistemas de software que comparten características similares. En este trabajo se desarrolló una LPS orientada hacia el campo de la inmótica, en la que se combinaron distintas tecnologías para automatizar la generación de aplicaciones, que incluyen: aplicación ejecutable y documentación. Algunas de las tecnologías que se integraron son: MDA (*Model-Driven Architecture*), Scala y AspectJ. El proceso de generación de las aplicaciones se realiza con base en un configurador.

Palabras clave: Variabilidad, *MDA*, inmótica, rasgos, aspectos.

Automated Generation of Inmotic Application with SPL Approach

Abstract: A software product line (SPL) refers to the methods of software engineering, tools and techniques used to create a set of software systems with similar characteristics. In this work was developed a SPL oriented to inmotic domain, with different technologies combined to automate the application generation, including: executable application and documentation. Some technologies that were integrated are: MDA (Model-Driven Architecture), Scala and AspectJ. The application generation process is based on a configurator.

Keywords: Variability, *MDA*, inmotic, traits, aspects.

1. Introducción

Hoy en día la competitividad entre las empresas que se dedican al desarrollo de software va en aumento, especialmente empresas que tienen como objetivo satisfacer las necesidades de un segmento de mercado. Estas empresas buscan cumplir con tres aspectos importantes que la ingeniería de software aun no logra del todo, como son: calidad, tiempo y costo. Por este motivo diversas empresas adquieren este paradigma como base de producción. Uno de los dominios que en la actualidad tienen una alta

demanda es el campo de la inmótica. Las empresas que se especializan en este campo requieren la implementación de un enfoque que permita garantizar la calidad de los productos y optimizar los beneficios que como empresa se requieren. Este trabajo integra distintas tecnologías para el desarrollo de una LPS, con el objetivo de automatizar la generación de aplicaciones y documentación específica para cada una de ellas. Las tecnologías que se consideran son: AspectJ, la Arquitectura Dirigida por Modelos (*Model-Driven Architecture*) y el lenguaje Scala, principalmente con el uso de *traits*.

La estructura general del artículo es la siguiente: en sección 2 se observa un resumen de los trabajos relacionados. En sección 3 se explica el enfoque propuesto. En sección 4 se detalla el caso de estudio en que se aplicó la propuesta. En sección 5 se muestran las conclusiones y trabajo a futuro. Finalmente las referencias.

2. Trabajos relacionados

En [1] se presentó la variabilidad mediante la integración del Desarrollo Dirigido por Modelos (*MDSD- Model Driven Software Development*) y Desarrollo de Software Orientado a Aspectos (*AOSD- Aspect-Oriented Software Development*). La utilización *MDSD-LPS* facilita la trazabilidad desde el dominio del problema hasta el dominio de la solución. El lenguaje orientado a aspectos es útil en la generación de código donde la arquitectura no proporciona enlaces. En [2] se propuso el método de Análisis Orientado a Aspectos (*AOA*) de los requisitos de los productos para el diseño de la Arquitectura de Línea de Productos (*PLA- Product Line Architecture*). El esquema de pasos del (*AOA*) es:(1) se separan los requisitos en cada aspecto de los requisitos originales, (2) se analizan los requisitos de cada aspecto por separado y se examina de forma aproximada la arquitectura requerida para cada aspecto, (3) se analizan los resultados y las opciones de diseño.

En [3] se presentó un enfoque que facilita la implementación, gestión y trazabilidad de la variabilidad mediante MDE (*Model-Driven Engineering*) y el Desarrollo de Software Orientado a Aspectos. Las características se separan en modelos y se componen por técnicas *AO* a nivel de modelo. Al integrar *MDSD* en SPLE (*Software Product Line Engineering*) los DSL (*Domain Specific Language*) manejan la variabilidad con respecto a su estructura o comportamiento. En [4] se presentó un nuevo enfoque para implementar LPS por mecanismos de reutilización de grano fino. Se introdujo el FRTJ (*Featherweight Record-Trait*), donde las unidades de funcionalidad de productos se modelan por los *traits* y los *records*. El grado de reutilización *traits-records* es más alto que el potencial de reutilización de las jerarquías que se basan en clases estáticas estándar.

En [5] se presentó una Arquitectura de Línea de Productos (*PLA- Product Line Architecture*). Esta investigación tuvo como objetivo obtener una *PLA* que: (1) cumpla con los atributos de calidad de los productos que se especifican, (2) que sea lo suficientemente genérico para generar los productos, (3) apoye las similitudes y variabilidad, (4) reúna los atributos de calidad para la LPS específica. En [6] se propuso un enfoque que facilita la gestión de la variabilidad en el modelado de la

arquitectura para la implementación de una LPS, donde los requerimientos del dominio, así como la arquitectura de la LPS se capturan en modelos. En la ingeniería de aplicación, el DSL se utilizó para especificar los requisitos de las aplicaciones concretas. Las técnicas *AO* se utilizaron durante la ingeniería de domino para modular los asuntos de interés en los modelos, transformadores y generadores.

En [7] se realizó un análisis que sirvió para identificar las debilidades de los Enfoques Orientados a Características (*FOA- Feature-Oriented Approach*), haciendo hincapié en la modularidad de la transversalidad. Se demostró que con la falta del apoyo de la modularidad transversal en los enfoques *FOA*, conduce a código disperso. Por otra parte, también se señaló que los *pointcut* como mecanismo para expresar la transversalidad en idiomas como AspectJ no es suficiente, ya que carece de soporte de módulos multiabstracción.

En [8] se habló sobre el montaje automatizado y la personalización de los componentes específicos del dominio de una *PLA*. Los niveles de abstracción en que se desarrolla la *PLA* son de bajo nivel. Para abordar este problema se utilizó la Ingeniería Dirigida por Modelos (*MDE*). Los beneficios obtenidos con *MDE* en LPS son: (1) agiliza el desarrollo de la *PLA* con la integración de las herramientas del modelado y la arquitectura de componentes de dominio específico, (2) las estructuras basadas en modelos ayudan a mantener la estabilidad de la evolución del dominio de los sistemas basados en *MDE*, (3) para mejorar la robustez y habilidad de la transformación de modelos se necesitan pruebas y soporte de depuración para ayudar a corregir y encontrar los errores en las especificaciones de transformación.

3. Propuesta

Para la construcción de la línea de productos se utilizó el *framework* de desarrollo para LPS [9]. Este *framework* distingue 2 procesos. El proceso de ingeniería de dominio se encarga de crear la plataforma con todos los artefactos que definen la parte común y la variabilidad de la LPS. El proceso de ingeniería de aplicación es el responsable de explotar la variabilidad que se definió y reutilizar los artefactos de la plataforma para obtener una aplicación específica.

La Arquitectura Dirigida por Modelos (*MDA-Model Driven Architecture*) se utilizó para definir una arquitectura que integra todos los elementos que trabajan en conjunto para el funcionamiento de una LPS. Esta integración no involucra o modifica los procesos que define el *framework* de desarrollo [9]. *MDA* permite la evolución de la LPS desde diferentes perspectivas, por lo que se diseñaron modelos en los que se seleccionaron 2 niveles de abstracción del enfoque, estos son: el Modelo Independiente de la Plataforma (*PIM-Platform-Independent Model*) y el Modelo

Específico de la Plataforma (*PSM-Platform-Specific Model*). En el modelo independiente se plantean conceptos a un alto nivel de abstracción, sin especificar las tecnologías que se van a utilizar. En el modelo de lado izquierdo de la figura 1, se observa el *PIM* con los siguientes elementos:

Dominio.- Corresponde al segmento de mercado para el que se desarrolla la LPS. En este caso, el *PIM* no determina hacia que segmento de mercado se orienta la LPS.

Modelo de características.- Para el desarrollo de la LPS existen diferentes lenguajes para modelar características. Esta definición permite a la LPS no depender de ningún lenguaje en específico.

Selección de características.- Este concepto se refiere a obtener una instancia del modelo de características que se utiliza, el cual representa las características de una aplicación.

Plataforma.- Contiene los artefactos que se obtuvieron al aplicar el *frame-work* de desarrollo [9]. Entre estos están los artefactos de requerimientos, diseño, implementación y pruebas, los cuales se diferencian entre artefactos del dominio y de aplicación, además se tienen los archivos que se encargan del ensamblaje de componentes de software.

Configurador.- Se encarga de utilizar de forma automática los artefactos de la plataforma, para verificar el dominio de la LPS y(o) generar aplicaciones.

Artefactos de dominio o aplicación.- Son los artefactos que se generan en función de los procesos que realiza el configurador.

A partir del *PIM* es posible obtener uno o más *PSM* que muestran una proyección del *PIM*. Para generar cada *PSM* se toman como base las reglas de transformación que se establecieron en el *PIM*. El *PSM* que se obtiene puntualiza las tecnologías que se utilizan para el desarrollo de una LPS. En el caso de estudio que se describe en la sección 4 se aplicó el *PSM* que se visualiza de lado derecho en la figura 1.

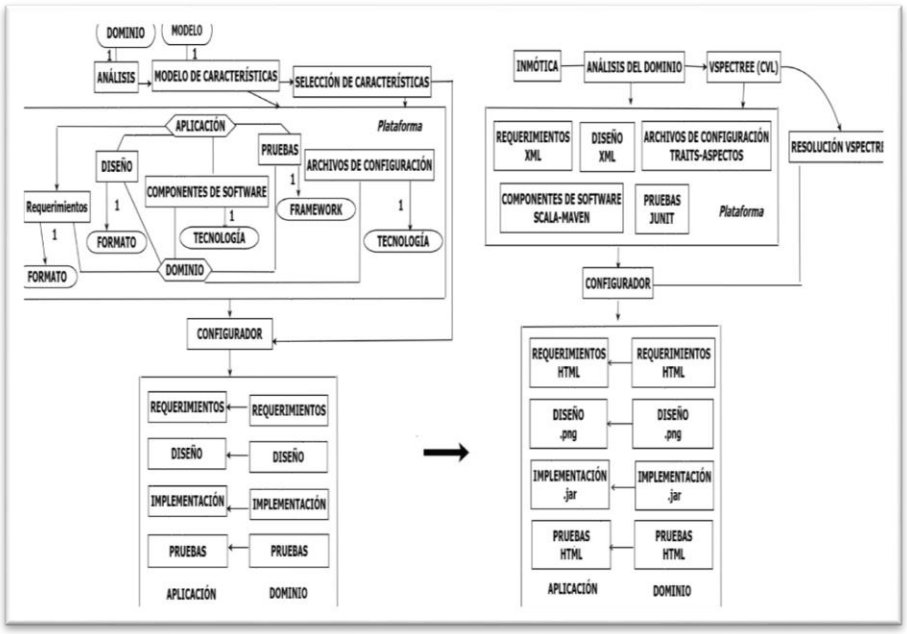


Fig. 1. *PIM* ***a*** *PSM.*

4. Caso de estudio

La inmótica es la incorporación de numerosos subsistemas en las instalaciones de uso terciario o industrial, con el fin de optimizar recursos, reducir costos y disminuir el consumo de energía innecesario, al mismo tiempo que aumenta la seguridad y el confort.

Los requerimientos que se obtuvieron para la LPS son de un caso real de una empresa privada. La administración de los productos de la LPS se realizó mediante un portafolio [9] que contiene la definición de 8 productos que la LPS es capaz de generar

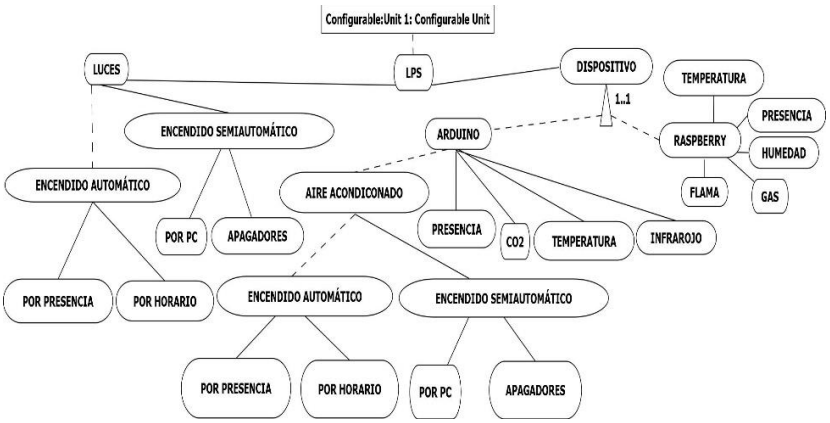


Fig. 2. Árbol de especificación de variabilidad.

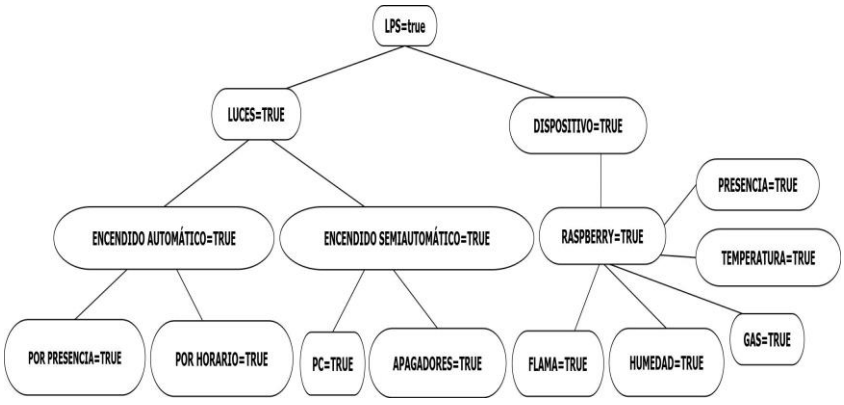


Fig. 3. Resolución de árbol de especificación de variabilidad para aplicación 7.

La variabilidad de la LPS se representó mediante un Árbol de Especificación de Variabilidad (*VSpectree*) con el Lenguaje Común de Variabilidad (*CVL-Common*

Variability Language) [10]. En la figura 2 se observa el *VSpectree* con las características de la LPS. Para representar las características de una aplicación con el *CVL*, es posible obtener instancias validas del árbol de figura 2. Al nuevo árbol que se obtiene se denomina Resolución de Árbol de Especificación de Variabilidad (Resolución *VSpectree*). A el árbol de la figura 2 que representa las características de una aplicación de la LPS, y se define con base en las restricciones del *CVL* [10]. En el árbol de figura 3 se observa la Resolución *VSpectree* para la aplicación o producto 7.

4.1. Configurador

El objetivo del configurador es automatizar los procesos de generación y verificación de artefactos de dominio y aplicación, con el fin de obtener aplicaciones completas que incluyan: documentación y aplicación ejecutable (.jar). El funcionamiento del configurador se basa principalmente en un archivo de configuración con formato XML y la selección de características válidas para una aplicación, como la resolución que se visualiza en la figura 3. La selección de características se realiza a través de la interfaz que se observa en la figura 4.

El archivo de configuración contiene la ruta donde se empaquetan las aplicaciones que se generan, además de 3 secciones que son: características, productos y recursos. En la sección de características se definen todas las características que administra la LPS, estas características son las que se encuentran en el árbol de especificación de variabilidad que se observa en la figura 2. La sección de productos describe todos los productos de la LPS con las características que conforman cada uno de ellos.

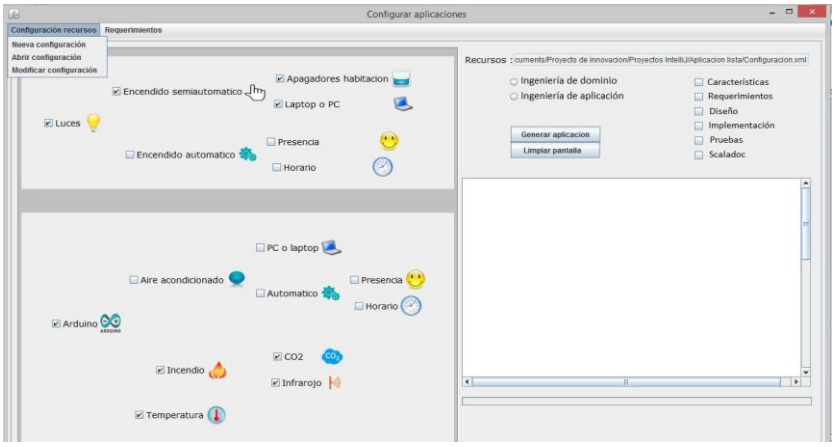


Fig. 4. Interfaz principal del configurador.

A continuación se muestra un ejemplo de la estructura del archivo de configuración en formato XML.

<lps nombre="LPS" ubicacion="C: /Aplicaciones/"> <caracteristicas>

```
    <caracteristica nombre="Luces por presencia"/> </caracteristicas>
  <productos>
    <producto nombre="Producto 1">
      <caracteristica nombre="Luces por presencia"/> </producto>
  </productos>
  <recursos>
     <recurso clave="1" tipo="Componentes_xml" url="C:/"/>
  </recursos>
</lps>
```

En el apartado que corresponde a recursos, se coloca la ubicación de los diferentes artefactos que se definieron con base en los procesos de ingeniería de dominio e ingeniería de aplicación del *framework* de desarrollo [9]. Una vez que se define el archivo de configuración es posible generar de forma automática las aplicaciones que se deseen.

4.2. Artefactos de la LPS

Para la definición de requerimientos textuales se diseñó un formato en XML donde se distinguen requerimientos de dominio y de aplicación, lo que permite al configurador (sección 4.1) obtener los requerimientos de forma automática.

A continuación se muestra un fragmento de la estructura del archivo XML para la definición de requerimientos.

```
<requerimientos>
<requerimiento clave="1" condicion="Base">R1</requerimiento>
<requerimiento clave="2" condicion="Flama">R3</requerimiento>
```

Un requerimiento que tiene el valor “Base” en el atributo condición, indica que el requerimiento pertenece al dominio, en caso de tener el nombre de alguna de las características que se describen en el diagrama de la figura 2, pertenecerá a la aplicación.

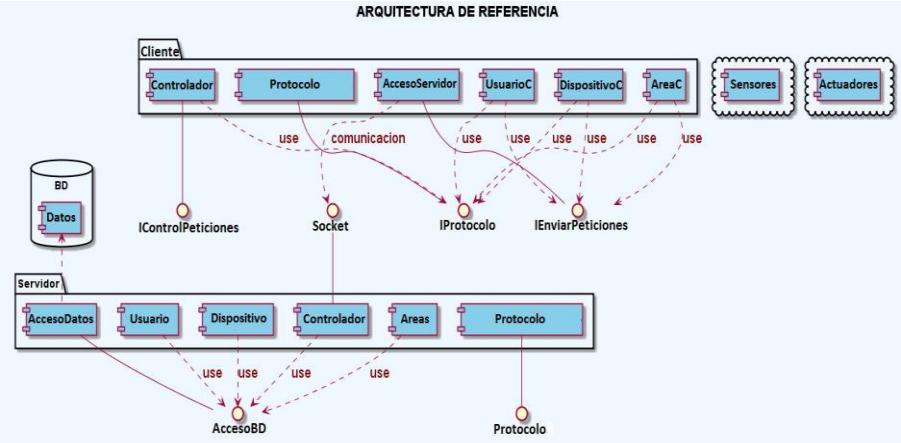


Fig. 5. Arquitectura de referencia.

En la fase de diseño de dominio y diseño de aplicación [9], se obtuvieron diferentes artefactos, entre ellos se encuentra la arquitectura de referencia que se reutiliza para cada una de las aplicaciones de la LPS, y la arquitectura de aplicación que representa la funcionalidad completa de una aplicación específica. En las figuras 5 y 6 se observa la arquitectura de referencia y únicamente la arquitectura del servidor de una de las aplicaciones de la LPS, donde se aprecia la reutilización de la arquitectura de referencia y el ensamblaje de los componentes del servidor necesarios para configurar la aplicación 7

Para automatizar la generación de los artefactos de diseño se definió un archivo XML que representa los componentes del dominio y de la aplicación. Este archivo facilita al configurador la selección de los componentes para generar el diagrama de la arquitectura de referencia y/o de la aplicación.

Los componentes que tienen el valor "Base" en el atributo condición, indican que el componente pertenece al dominio, en caso de tener el nombre de una característica, pertenece a la aplicación. La sintaxis que se utilizó se basa en la herramienta PlantUML [11]. A continuación se muestra un fragmento de la estructura del formato XML para la definición de componentes.

```
<componentes>
  <componente clave="1" condicion="Base"> [C1] </componente> <componente clave="2"
  condicion="Luces"> [C2] </componente>
```

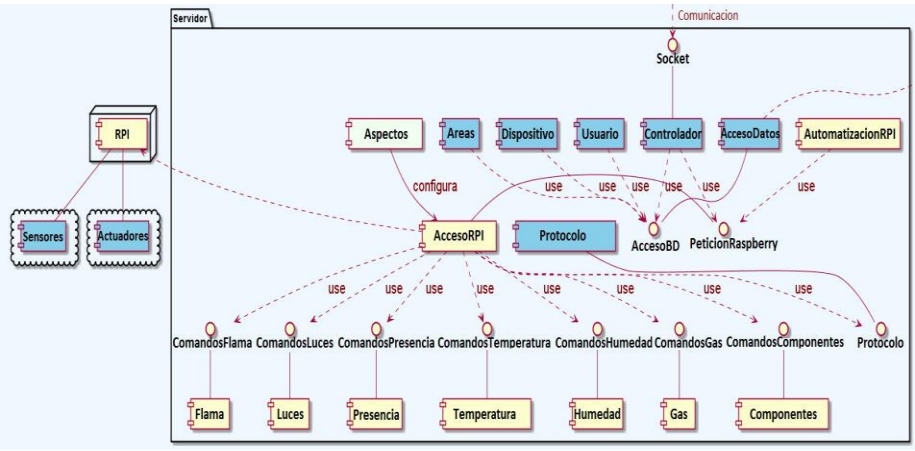


Fig. 6. Arquitectura del servidor para aplicación 7.

Durante la fase de realización, el configurador genera de forma automática la aplicación (ejecutable.jar) con la herramienta *Apache Maven* [12]. La aplicación se configura con base en la arquitectura de aplicación que se obtiene en la fase de diseño de la aplicación. El ensamblaje de las diferentes variantes para cada aplicación se realiza con composición *mixin* mediante el uso de *traits* del lenguaje Scala.

Cada *trait* que se utiliza contiene la funcionalidad para las principales características que se definieron en el *VSpectree* (Figura 2), de tal forma que la

configuración de características para cada aplicación basta con agregar o quitar *traits* en la composición *mixin*. El elemento que hereda el comportamiento de los *traits* es otro *trait* (CaracteristicasAplicacion). Además de la configuración con composición *mixin*, también se utilizaron aspectos para conectar componentes a nivel binario.

En la figura 7 se observa una parte de un diagrama de clases de una aplicación con la combinación de aspectos y *traits*. La fase de pruebas requiere de la entrega de los artefactos que proporciona la fase de realización de la aplicación, que corresponde a los componentes de software con la configuración de una aplicación específica.

Para la reutilización y manejo de la variabilidad de las pruebas se definieron *suites* de pruebas con el *framework JUnit*. Estas *suites* se ejecutan de forma automática para verificar únicamente el dominio de la LPS o la aplicación completa que se va a generar.

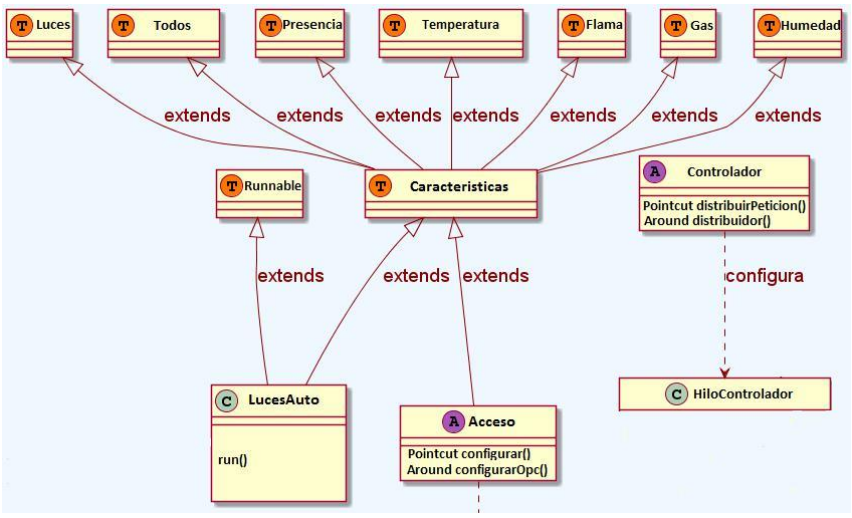


Fig. 7. Combinación de aspectos y *traits.*

Cada una de las clases que se especifican en la suite contiene las diferentes pruebas para el dominio y las aplicaciones. A continuación se muestra un ejemplo de Suite de pruebas para la aplicación 7.

```
@RunWith (classOf [Categories])
@IncludeCategory (classOf[Producto7])
@SuiteClasses(Array(classOf[Luces],classOf[Automatizacion)) class
SuiteProducto7Test {}
```

En la implementación para cada prueba de la plataforma se determina si pertenece al dominio, para una aplicación específica o para ambos. De esta forma se maneja la variabilidad de forma automática para evitar aplicar pruebas que no corresponden a

la aplicación que se desea generar. Si la prueba no devuelve el valor que se establece, se marca como error y se registra automáticamente en un reporte de pruebas. Se tienen 75 pruebas en la LPS.

A continuación se muestra un ejemplo de la estructura de una prueba para las aplicaciones 7 y 8.

```
@Category(Array(classOf[Producto7],classOf[Producto8]))
@Test
def pruebaDeIntegracionEstadoDeSensorPresenciaRaspberry={}
```

5. Conclusiones y trabajo a futuro

La automatización de los procesos de la LPS disminuyó el tiempo de generación de aplicaciones y mejoró la calidad por la constante revisión de los componentes de software que se realizan en una LPS, además de aplicar el esquema de pruebas que maneja el *framework* de desarrollo. El uso de *MDA* permitió separar las responsabilidades de los diferentes elementos que integran la LPS en un *PIM* y *PSM*, lo que facilita la evolución de la LPS en ambos modelos. Con la combinación de Scala y AspectJ se obtuvieron beneficios para el manejo de la variabilidad en la implementación de la LPS, con el uso de *traits* mediante composición *mixin* y con aspectos para el ensamblaje de componentes de software a nivel binario. El desarrollo del configurador permitió automatizar la generación de aplicaciones en las fases de requerimientos, diseño, implementación y pruebas. Como trabajo a futuro se pretende incrementar el número de aplicaciones del portafolio de productos de la LPS, agregar más tipos de diagramas UML, y aumentar la capacidad del configurador para crear más de una aplicación de forma concurrente.

Referencias

1. Voelter, M., Groher, I.: Product line implementation using aspect-oriented and model-driven software development. In: Software Product Line Conference, SPLC 2007, 11th International, IEEE, pp. 233–242 (2007)
2. Kishi, T., Noda, N.: Aspect-oriented analysis for product line architecture. In: Software Product Lines, Springer, pp. 135–145 (2000)
3. Groher, I., Voelter, M.: Aspect-oriented model-driven software product line engineering. Transactions on aspect-oriented software development VI, Springer, pp. 111–152 (2009)
4. Bettini, L., Damiani, F., Schaefer, I.: Implementing software product lines using traits. In: Proceedings of the 2010 ACM Symposium on Applied Computing, ACM, pp. 2096–2102 (2010)
5. Verdín, M. K., Olalde, C. L., Van Der Hoek, A., Peña, J. F.: Diseño de Arquitecturas de Líneas de Productos de Software empleando AOPLA. In: VIII CIINDET, Cuernavaca, Morelos, México (2009)
6. Groher, I., Voelter, M., Schwanninger, C.: Integrating Models and Aspects into Product Line Engineering. In: SPLC, pp. 355 (2008)

7. Mezini, M., Ostermann, K.: Variability management with feature-oriented programming and aspects. In: ACM SIGSOFT Software Engineering Notes, ACM, Vol.29, pp. 127–136 (2004)
8. Deng, G., Schmidt, D. C., Gokhale, A., Gray, J., Lin, Y., Lenz, G.: Supporting Evolution in Model-Driven Software Product-line Architectures. In: ACM SIGSOFT Software Engineering Notes, ACM (2007)
9. Van Der Linden, F., Pohl, K.: Software Product Line Engineering: Foundations, Principles, and Techniques. In: Springer Science Business Media (2005)
10. CVL Submission Team: Common variability language (CVL), OMG revised submission (2012)
11. PlantUML, http://plantuml.com/
12. Maven, I.: Apache maven project (2011)

ISSN 1870-4069

Propuesta de notación para el modelado de elementos de programación funcional en UML

Nanny Rodríguez-Munguía, Ulises Juárez-Martínez,
Gustavo Peláez-Camarena, Alma Sánchez-García

nrodriguezmunguia@acm.org, ujuarez@ito-depi.edu.mx,
sgpelaez@yahoo.com.mx, alivsaga@hotmail.com

Resumen. El paradigma de programación Objeto Funcional se ha extendido en diversos lenguajes de programación como son Scala, Java, C#, Ruby, entre otros; con ello surge la necesidad de que las herramientas de desarrollo permitan definir cómo abordar los nuevos conceptos en la codificación. Un ejemplo claro es UML, el cual tiene los medios necesarios para modelar los diversos contenidos del desarrollo de software, gracias a sus mecanismos de extensión; sin embargo, no posee un estándar para la modelación de los elementos de la programación Funcional, es por ello que en el presente trabajo se introduce una propuesta que permita definir el uso de elementos y propiedades funcionales, en el desarrollo de software con lenguajes Objeto Funcionales, como son: funciones de orden superior, funciones *currificadas*, evaluación perezosa, lambdas y mónadas.

Palabras clave: Programación funcional, programación orientada a objetos, lenguajes híbridos, paradigma orientado a objetos funcionales, Scala, desarrollo de software.

Propose of Notation for Functional Programming Modeling in UML

Abstract The Object-Functional programming paradigm it has been extended in various programming languages such as Scala, Java, C#, Ruby, among others. With this arise the necessity that the development tool allow to define how to approach the new concepts in the codification. A clear example is UML, which has the necessary means to model the diverse contents of software development, thanks to its extension mechanisms. However, it does not have a standard for the modeling of functional programming elements. But this is why presents in this work a proposal that allows define the use of properties and elements of the functional programming in Oriented Object-Functional languages, as are: high order functions, curried functions, laziness evaluation, lambdas and monads.

Keywords: Functional programming, oriented object programming, hybrid languages, paradigm oriented to functional objects, Scala, software development.

1. Introducción

En años recientes los lenguajes Orientado a Objetos, como Java y C#, han adoptado el paradigma de programación Funcional para brindar una mayor variedad de soluciones, que no eran posibles en este enfoque. Mientras que otros lenguajes, como Scala, han surgido considerando ambos paradigmas de programación desde su concepción.

En la actualidad el paradigma OO tiene amplio soporte para su modelado en UML [1], el cual no sólo proporciona los elementos necesarios para el modelado OO, sino que también posee mecanismos de extensibilidad que le permite modelar otros elementos que se integran dentro de un proyecto de software, como scripts, módulos, subsistemas y sistemas. Por otro lado el enfoque funcional no ha presentado madurez en el modelado de sus elementos; en este paradigma se identificó la metodología FAD [2], la cual proporciona elementos para el modelado de propiedades funcionales, pero no cumple con todas ellas, por lo que su soporte es limitado.

Aún con el amplio soporte de modelación OO que tiene UML, carece de estándares para el modelado de elementos funcionales que se encuentran inmersos en el enfoque de objetos. Es por esto que se propone una notación que sirva como estándar en el uso de propiedades funcionales en el lenguaje de modelado UML.

En el estado de la práctica se identificó que UML posee un amplio uso para la representación de los diversos enfoques que convergen en el desarrollo de proyectos de software: desde la modelación de elementos de hipermedia, hasta los nuevos paradigmas que mejoran la encapsulación de requerimientos no funcionales como lo es la programación Orientada a Aspectos. Esto demuestra que UML posee la capacidad ampliarse; sin embargo, sin un estándar, esta capacidad sólo le sirve a quienes adoptan su propia notación, como empresas, grupos e investigadores de área, por lo que limita la capacidad de compartir los modelos entre los diferentes grupos.

Al crear una notación estándar permitirá la generación de modelos que expresen las prácticas en el paradigma de la programación Objeto Funcional, con lo cual podrán abordarse independientemente del lenguaje de programación y así mejorar la difusión de soluciones.

2. Trabajos relacionados

Para el desarrollo de la propuesta, se realizó la investigación sobre los elementos y propiedades funcionales, así como propuestas de notación utilizando UML para diversos enfoques. A continuación se presentan los trabajos más relevantes.

En [2] se presentó la metodología de Análisis y Diseño Funcional (FAD) (del inglés Functional Analysis and Design), junto con una notación gráfica de elementos funcionales y sus relaciones. El soporte que brinda a nivel notación permite la modelación de tipos, herencia de tipos, funciones de orden superior, dependencia de funciones, funciones currificadas, módulos, subsistemas y sistemas; sin embargo, no aborda el uso de lambdas, mónadas ni evaluación perezosa. La metodología FAD abarca en su proceso la mejora de funciones, esto a través de técnicas que se basan en

la identificación de firmas de funciones durante el modelado, de manera que maximice la reutilización.

En [3] se aborda el uso de los mecanismos de UML para ampliar el lenguaje, de manera que permita el modelado de elementos ajenos al enfoque OO. En este trabajo se menciona que el mecanismo más utilizado es el estereotipo, por lo que se ha presentado un abuso en su utilización, y sugiere algunas medidas para mejorar su uso.
En [4] se describió la necesidad de las extensiones de UML, para el modelado de elementos de hipermedia utilizados en el desarrollo de software para la World Wide Web, de manera que brinde el soporte para crear estructuras de navegación. En este trabajo se presentó una propuesta para el diseño de hipermedia a través de uso de estereotipos, el uso del Object Constraint Language (OCL), mecanismos de extensión de UML.

En [5] se presenta una propuesta para el desarrollo de software utilizando el enfoque de la programación Orientado a Aspectos (AOP), utilizando UML y sus mecanismos de extensión para representar los asuntos de corte. Esta propuesta también se apoyó en el uso de estereotipos para describir los diferentes tipos de aspectos y el modelado del entrelazado de código. Este trabajo muestra que UML es capaz de extenderse para representar enfoques de programación que se integran en el paradigma OO.

El estado de la práctica indica que FAD no tiene el soporte suficiente para integrarse con el Modelado de Objetos, mientras que UML posee múltiples muestras de su capacidad para abordar los diferentes enfoques en el desarrollo de software. Por lo tanto, es factible realizar una propuesta que permita integrar los elementos de la programación Funcional, utilizando los diferentes mecanismos de UML y su estructura.

3. Notación propuesta

El Enfoque Funcional posee diversas propiedades, algunas no son exclusivas del paradigma (como el polimorfismo paramétrico), mientras que otras se encuentran inmersas en el lenguaje de programación (como es la tipificación fuerte y ciudadanos de primera clase); la propuesta que aquí se presenta, permitirá representar los elementos funcionales como son funciones de orden superior, funciones currificadas, evaluación perezosa, mónadas y el uso de lambdas; de manera que se introduzcan en las fases de modelado de software y finalmente su codificación. Los ejemplos que se muestran a continuación presentan su codificación en lenguaje de programación Scala.

3.1. Funciones de orden superior

Las funciones de orden superior son aquellas que tienen la capacidad de recibir una o más funciones como argumentos, o bien, devuelven una función como valor de retorno; esto recae en la necesidad de representar este tipo de argumento poco usual: una operación (método o función). Para el Enfoque Funcional, todas las funciones poseen una firma [2], comúnmente representada a través del símbolo -> que separa cada argumento y el último indica el valor de retorno. Para el enfoque OO, en la notación UML [1], una operación se define a través de un nombre seguido de un paréntesis

dentro de los cuales se presentan los argumentos que recibirá con un nombre y su tipo separado por dos puntos, cada argumento separado por comas, junto a los paréntesis se usan dos puntos y el tipo de retorno.

Dado que en la notación de UML no se requiere especificar un nombre para los argumentos, la notación propuesta es: f(type, type, ..)->type , donde f es un alias para la función, entre paréntesis los tipos de datos que recibe, separados por comas seguido del símbolo ->, el cual señala al tipo de retorno. En la figura 1 se muestra un ejemplo, donde operation1 recibe una función f que tiene como parámetros un valor de tipo int y devuelve otro valor de tipo int; mientras que operation2 recibe un parámetro de tipo boolean y devolverá una función f cuyos parámetros son: el primero de tipo String y el segundo de tipo int, devolviendo un tipo double.

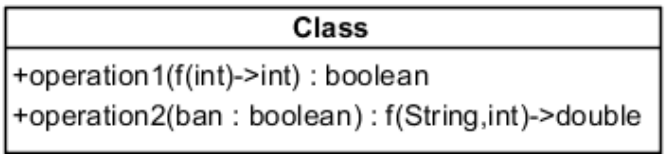


Fig. 1. Ejemplo de notación para funciones de orden superior.

La implementación en código del ejemplo de funciones de orden superior es el que se muestra a continuación en el algoritmo 1, obsérvese que dentro del método operation2 se utilizan funciones al vuelo (lambdas), las cuales se revisan más adelante.

Algoritmo 1. Implementación en código de la clase Class y los métodos operation1 y operation2.

```
class Class{
  def opertation1(f:(Int)=>Int):Boolean={
    return f(8)>100;
  }

  def operation2(ban:Boolean):(String, Int)=>Double={
    if(ban)
      return ((a:String, b:Int)=>a.length()/b);
    else
      return ((a:String, b:Int)=>a.length()*b);
  }
}
```

Esta notación permite claridad en la separación de los tipos que son argumentos, del tipo de valor de retorno, aunque esta notación casi coincide con la sintaxis usada en el lenguaje Scala, para definir una función como argumento de otra función, se diferencia únicamente por el símbolo -> por =>. En el caso de una función como valor de retorno, es posible omitir el alias, puesto que puede usarse de manera anónima o designarle un nuevo nombre dependiendo de su implementación.

3.2. Funciones currificadas

Una función se denomina currificada, cuando recibe sus parámetros uno a la vez; en los lenguajes OO esto no ocurre de manera común, solo algunos lenguajes soportan esta propiedad. En Scala [7] es posible definir una función currificada de forma variable, es decir, cuántos y qué parámetros recibirá en cada llamada parcial, por ello es necesario especificar la distribución de los argumentos y el orden de las posibles llamadas parciales; la notación propuesta para esta propiedad es definir la separación de argumentos con paréntesis, de manera que se separen estas llamadas parciales. Como se muestra en la figura 2, se tiene al método operation1(a:int , b:int, c:int)(d:int, e:int)(f:int):boolean, el cual en su primera llamada parcial debe contener tres argumentos de tipo int, esto retornará una operación anónima que recibe tres parámetros de tipo int, que de igual forma es posible llamar parcialmente, enviando dos parámetros de tipo int, retornando otra función anónima que tendrá como parámetro un tipo int, al llamar esta última función, con el parámetro faltante, se completará la llamada y retornará el resultado de tipo double. En los otros ejemplos operation2 y operation3, se muestra que la distribución de currificacion se puede especificar de forma variable, agrupando argumentos de uno en uno o más. Nótese que operation3 recibe como argumento currificado una función f(int)->boolean; de hecho las funciones currificadas también son funciones de orden superior, con el simple hecho de retornar funciones, y ahora también al recibirlas como argumento. En el algoritmo 2 se muestra el código correspondiente a este ejemplo.

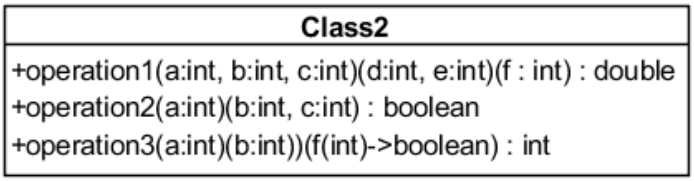


Fig. 2. Ejemplo de notación para funciones currificadas.

Algoritmo 2. Implementación en código del modelo de la clase Class2 y sus respectivos métodos.

```
class Class2 {
  def operation1(a:Int, b:Int, c:Int)(d:Int, e:Int)(f:Int):  Double = {
        return a + b + c / (d * e) - f;
    }
  def operation2(a:Int)(b:Int, c:Int): Boolean = {
        return a > (b - c)
    }
  def operation3(a:Int)(b:Int)(f:(Int) => Boolean): Int ={
        if (f(a))
            return a + b;
   else
```

```
            return a - b;
  }
}
```

A pesar de que las llamadas parciales retornan funciones, no es necesario especificar esto, puesto que se sobreentiende que al aplicar la currificación con llamadas parciales, la función devuelta será la misma pero con los argumentos faltantes para completar la llamada a dicha operación.

3.3. Evaluación perezosa

La evaluación perezosa permite que los valores se evalúen únicamente cuando son necesarios, esta estrategia permite la manipulación de estructuras de datos infinitas, denominadas flujos (*streams*). Dado que la evaluación perezosa se aplica sobre valores, la notación que se propone debe aplicar sobre los atributos de una clase.

UML tiene soporte para el nivel de visibilidad de dichos atributos, entre los cuales se hace uso de los símbolos # (protegido), - (privado), + (público) y ~ (paquete), por lo que estos símbolos se descartan como opciones de notación. Sin embargo, el símbolo "~" no se usa frecuentemente, y es uno de los símbolos ASCII que se asocia fácilmente a la palabra flujo, por ello la notación propuesta es hacer uso de los símbolos :~ de forma conjunta donde ":" significa evaluación y "~" significa flujo, dando a entender que dicho atributo se evaluará de forma que mejore su desempeño, esto da apertura al uso de la evaluación perezosa.

En la figura 3 se muestra la clase InfineCalculus, la cual contiene dos métodos de cálculo infinito fibonacci() y count(), la clase LazzinesEvaluation, posee 3 atributos que serán evaluados de forma perezosa, fibonacciResults (es público), numbers (es visibilidad de paquete), y x (privado); los dos primeros indican que serán variables auxiliares en la obtención del cálculo fibonacci() y de count().

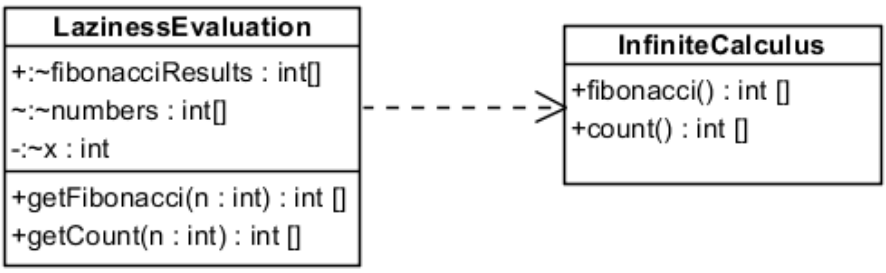


Fig. 3. Ejemplo de notación para la evaluación perezosa.

Nótese que el atributo x, también se evalúa de manera perezosa, sin embargo, no es un arreglo de int, esto es debido a que no contendrá resultados de los métodos de InfiniteCalculus, mostrándose que cualquier atributo tiene la capacidad de evaluarse de forma perezosa, y no es exclusivo de una estructura de datos. En el algoritmo 3 se muestra la implementación en código de Scala, del modelo de ejemplo para la

evaluación perezosa. En este ejemplo se demuestra que la notación propuesta no se confunde con la de visibilidad de paquete, estándar de UML.

Algoritmo 3. Implementación en código del modelo presentado en la fig. 3 para la evaluación perezosa de cálculos infinitos.

```
class InfiniteCalculus {
  def count(): Stream[Int] = {
    Stream.cons(0, count() map (_ + 1));
    }
  def fibonacci(): Stream[Int] = {
    Stream.cons(0, (0 #:: fibonacci.scanLeft(1)(_ + _)));
    }
}
class LazinessEvaluation{
  val cal=new InfiniteCalculus();
  lazy val fibonacciResults=cal.fibonacci();
  lazy protected val numbers=cal.count();
  lazy private val x={println("Hola");100}

  def getFibonacci(n:Int):Array[Int]={
    fibonacciResults.take(n).toArray
  }
  def getCount(n:Int):Array[Int]={
    numbers.take(n).toArray
  }
}
```

3.4. Mónadas

De acuerdo con [8], las mónadas son un mecanismo de la programación funcional que permite la introducción de declaraciones imperativas y proporciona una manera de abstraer sobre diferentes tipos de cálculos. Un cálculo se considera como una función que típicamente producirá un valor, cada cálculo se caracteriza por una estructura específica de los parámetros y valores de retorno, al definir un constructor de tipos, se define el tipo de este cálculo. Un ejemplo es la mónada *Maybe*, que define cálculos que se caracterizan por producir un valor, pero es posible que falle, los tipos de esta clase de cálculo se define como *Just* y *Nothing*, donde *Just* es el valor producido, y *Nothing* es el caso de fallo.

Dada esta definición, los cálculos de este tipo pueden ser unidos secuencialmente, el valor se inyecta extrayéndolo de su contexto actual y enlazándolo a otra función que acepta dicho valor. Para realizar el enlace es necearia una función que permita mapear el valor actual de la mónada hacia otra función del mismo tipo, que en los lenguajes funcionales es denominado *bind***.** Dada esta definición, un tipo monádico, en lenguajes Objeto Funcionales, es una clase que posee la encapsulación de un valor en cierto

contexto, y tiene el comportamiento necesario para mapear o enlazar dicho valor fuera de ese contexto, de manera que se crea un nuevo objeto perteneciente al mismo tipo.

La representación que se muestra en la figura 4 es la mónada *Maybe*. Este modelado se apoya en la notación propuesta para funciones de orden superior para definir el método que funcionará como el enlace, en este caso map(); la notación estándar de UML para representar jerarquía, que representa las dos posibles construcciones de Maybe; y clases genéricas para la referencia de un valor en dado contexto. El código resultado de este modelo se muestra en el algoritmo 4, cabe destacar que para la definición de una estructura monádica, los lenguajes se apoyan de diferentes características del propio lenguaje, como en Scala es el uso de *sealed trait*, *object*, y *case*.

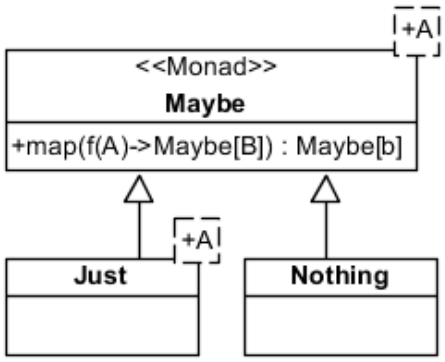


Fig. 4. Ejemplo de notación para una mónada.

Algoritmo 4. Código fuente correspondiente a la mónada Maybe.

```
sealed trait Maybe[+A] {
  def map[B](f: A => Maybe[B]): Maybe[B] = this match {
    case Just(a) => f(a)
    case Nothing => Nothing
  }
}
case class Just[+A](a: A) extends Maybe[A] {
  override def map[B](f: A => Maybe[B]) = f(a)
}
case object Nothing extends Maybe[Nothing] {
  override def map[B](f: Nothing => Maybe[B]) = Nothing
}
```

La definición que se muestra en la figura 4 requiere un mayor nivel de detalle; por otro lado, no se asevera que todas las mónadas se apoyen del uso de genéricos, además de que la especificación del mapeo variará en función a las necesidades. Por ello se propone el uso del estereotipo <<Monad>>, en adición a la representación de la mónada, de esta manera, se hace referencia a este tipo de construcción de forma más directa; el uso de genéricos y la definición de métodos para el mapeo, será parte del

refinamiento de dicho modelo. En la figura 4 se muestra el uso del estereotipo propuesto.

3.5. Uso de lambdas

Una de las propiedades funcionales cuya notación, a nivel de especificación de UML, varía dependiendo de la forma en que se implementa, son las expresiones lambda. Las expresiones lambda son funciones anónimas que se implementan cuando son necesarias, estas se utilizan comúnmente cuando se envían como parámetro a otras funciones. Definir el uso de funciones anónimas en un modelo UML implica describir el comportamiento interno de un método, sin embargo, ese nivel de descripción no es común.

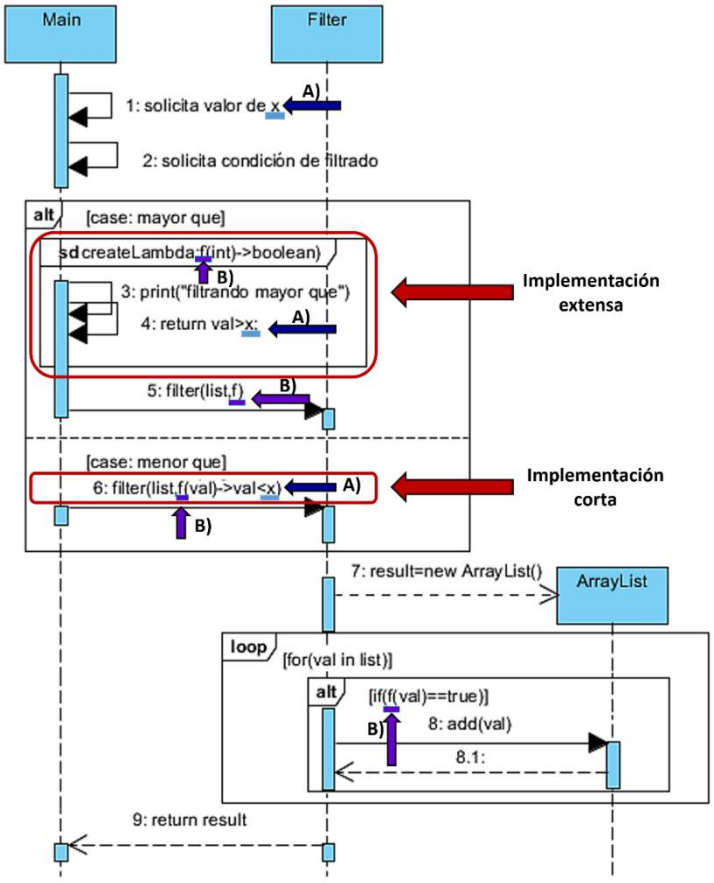


Fig. 5. Ejemplo de modelado para el uso de lambdas.

Una forma factible para describir la creación de una función anónima es a través de un diagrama de interacción, donde se observa la llamada y los argumentos que se envían

a un método, por lo que se hace referencia al comportamiento dinámico del sistema. Esta forma de modelación es especialmente útil si existen diferentes versiones de dicha función en casos alternos.

En la figura 5 se muestra un caso particular, donde la clase Filter posee una función de orden superior, llamada filter(), la cual recibe dos parámetros: una lista y una función; y realiza un filtrado de dicha lista a partir de la función recibida. La clase Main solicita un valor para x, y el tipo de filtrado a realizar, de manera que, de acuerdo a la selección, se creará una función anónima para cumplir dicho requisito.

En el caso de implementación extensa, se hace uso de un elemento *frame* de UML, en el cual se especifica createLambda:f(int)->boolean, donde se hace referencia a la creación de una lambda cuya firma de método es la señalada después de dos puntos.

Este *frame* es necesario, debido a que una implementación extensa requiere más de una instrucción; en este caso, se imprime "filtrando mayor que", cada vez que se hace la llamada al método, y la última instrucción señala el valor de retorno, que es la evaluación de mayor qué entre val (un alias del valor a recibir) y el valor de x (un valor ya definido en el objeto actual, inciso A). Finalmente se hace la llamada a la función de orden superior filter(), al cual se le envía como parámetro el alias de la función definida en el *frame* (ver inciso B).

En el caso de la implementación corta, simplemente se hace la llamada de la función filter(), enviando la lista y de forma breve la función anónima, de forma similar a la firma del método, señalando el alias de la función, seguido de paréntesis (dentro de los cuales se definen los parámetros a recibir) , el símbolo -> que señala a la instrucción a realizar, en este caso la evaluación menor qué entre val y x.

Es importante mantener la consistencia de los alias de función en el diagrama, puesto que de no hacerlo dificultará su interpretación. Por otro lado, en la implementación no es forzoso mantener dicha consistencia, sin embargo, es aconsejable para facilitar la lectura del código.

En el algoritmo 5 se muestra cómo es la implementación del ejemplo mostrado en el diagrama de la fig. 5, lo que demuestra que el modelo es factible para la programación en el lenguaje Scala, dado que el uso de lambdas es trasparente para el programador (todos los lambdas implementan de la clase Lambda); por lo que la base para su implementación es definir su firma y su proceso interno. Sin embargo en el lenguaje Java (versión 8), el uso de lambdas es a través de interfaces funcionales; por lo que es necesario señalar la interfaz funcional que implementa el lambda que se define; esta especificación es parte del refinamiento del modelo, por lo que se parte inicialmente con la definición del lambda en el diagrama de secuencia, y posteriormente crear las interfaces funcionales necesarias, para cumplir con las firmas de función.

Algoritmo 5. Código correspondiente al diagrama de secuencia de la fig. 5.

```
class Filter {
  def filter(list: List[Int], f: (Int) => Boolean): ArrayList [Int] = {
    var newList = new ArrayList[Int]();
    for (valor <- list) {
      if (f(valor) == true)newList.add(valor)
```

```
    }
    return newList;
  }
}
object Main {
  def main(args: Array[String]) {
    val x = readLine().toInt
    var list = List(1, 2, 3, 4, 5, 6, 8);
    print("introduce condición de filtrado");
    readLine() match {
      case "mayor que" => { //implementación larga
        def f = (valor: Int) => {
          print("filtrando mayor qué")
          (valor > x)
        }
        new Filter().filter(list, f);
      }
      case "menor que" => new Filter().filter(list, (valor => valor < x)); //implementación corta
    }
  }
}
```

4. Conclusiones y trabajo futuro

La notación propuesta para representar elementos funcionales es completamente compatible con UML, dado que se apoya de los elementos que proporciona este lenguaje de modelado, y también se asocia a lo nomenclatura utilizada en el enfoque funcional. Los ejemplos utilizados demuestran que es viable utilizar esta notación en aplicaciones reales, puesto que todos los ejemplos fueron realizados en *Visual Paradigm*, una herramienta para el modelado con UML.

Como trabajo a futuro se espera evaluar la notación a través del desarrollo de proyectos en los distintos Lenguajes Objeto Funcionales, de manera que no presente inconsistencias al realizar la codificación.

Referencias

1. Rumbaugh, J., Jacobson, I., Booch, G.: El Lenguaje Unificado De Modelado, Manual De Referencia. Segunda edición Madrid, Pearson Educación, S.A (2007)
2. Russell, D. J.: A Functional Analysis and Design Methodology. Ph.D. dissertation, Universidad de Kent en Canterbury (2000)
3. Baumeister, H., Koch, N., Mandel, L.: Towards a UML extension for hypermedia design. In: International Conference on the Unified Modeling Language, Springer Berlin Heidelberg, pp. 614–629 (1999)

4. Henderson-Sellers, B., Gonzalez-Perez, C.: Uses and abuses of the stereotype mechanism in UML 1.x and 2.0. In: International Conference on Model Driven Engineering Languages and Systems, Springer Berlin Heidelberg, pp. 16–26 (2006)
5. Chiusano, P., Bjarnason, R.: Functional Programming in Scala. 1st ed, Manning Pubications Co. (2012)
6. Ballal, R., Hoffman, M. A.: Extending UML for aspect oriented software modeling. In: Computer Science and Information Engineering, 2009 WRI World Congress on, IEEE, Vol. 7, pp. 488–492 (2009)
7. Hoefer, C., Ostermann, K.: On the relation of aspects and monads. In: Sixth International Workshop on Foundations of Aspect-Oriented Languages, FOAL (2007)

ISSN 1870-4069

Programación e implementación del software de aplicación “Fácil Hogar” para aminorar el índice de desperdicio de alimentos en los hogares de Tlaxcala

Amado Sánchez, Josefina Angelino, Daniel Temoltzin, Arturo Águila

Universidad Politécnica de Tlaxcala, Hueyotlipan,
Tlaxcala, México

{amado.sanchez1982, josefinaaz12}@gmail.com,
{dtvega2010, ing.arturo_aguila}@hotmail.com

Resumen. Datos del Banco Mundial reflejan que entre un cuarto y un tercio de los alimentos producidos anualmente para consumo humano a nivel mundial se desperdician; en México se pierden 10 millones de toneladas de alimentos por caducidad. Las principales causas provienen del comportamiento del consumidor teniendo un elevado índice de pérdidas en el hogar equivalente a un 28%. Se presenta el diseño del Software de aplicación “Fácil Hogar" dirigido a amas de casa de zonas urbanas en Tlaxcala, creado en la plataforma *APP INVENTOR 2*, que permite la administración de entradas y salidas de los alimentos adquiridos. El presente proyecto propone contrarrestar los desperdicios generados en el hogar, controlando las entradas y salidas de la alacena y el refrigerador, indicándole a la ama de casa el estado del alimento en base a su fecha de caducidad para que sepa en todo momento que productos debe consumir en primera instancia. Dentro de los resultados alcanzados se obtuvo una mejora en disminuir el desperdicio de alimentos y costos en el hogar.

Palabras clave: Aplicación, App Inventor, alimentos, desperdicio, reducción.

Programming and Implementation of the "Easy Home" Application Software to Reduce the Food Waste Index in Tlaxcala Households

Abstract. World Bank data show that between one-quarter and one-third of the food produced for human consumption worldwide is wasted annually; in Mexico 10 mil-lion tons of food per expiry are lost. The main causes are consumer behavior, with a high loss rate in the household of 28%. The design of the "Easy Home" application software for urban housewives in Tlaxcala, created in the APP INVENTOR 2 platform, which allows the administration of inputs and outputs of purchased foods, is presented. Counteract the waste generated in the home, controlling the entrances and exits of the cupboard and the refrigerator, indicating to the housewife the state of the food based on its expiration date so that it knows

at all times which products to consume in the first instance Within the results obtained, an improvement was obtained in reducing food waste and costs in the household.

Keywords: Application, App Inventor, food, waste, reduction.

1. Introducción

El objetivo del presente proyecto se basa en desarrollar una aplicación para dispositivos *Android* que permita a las amas de casa contrarrestar los desperdicios generados en el hogar, controlando las entradas y salidas de la alacena y el refrigerador, indicándole a la ama de casa el estado del alimento en base a su fecha de caducidad para que sepa en todo momento que productos debe consumir en primera instancia.

Para llevar a cabo el desarrollo de la aplicación se debe tener en cuenta que dentro del mercado de dispositivos móviles, los que están experimentando un mayor crecimiento de ventas son los *tablets* y los *smartphones*. Estos últimos serán los utilizados para poder crear la aplicación.

Actualmente, alrededor del 53.4 millones de dispositivos móviles existen en México, lo que permite una mayor expansión en el uso de la aplicación desarrollada.

1.1. Motivación del desarrollo de la aplicación

Aproximadamente 870 millones de personas pasan hambre todos los días, mientras que la asombrosa cantidad desperdicio de alimentos en el mundo ha crecido exponencialmente. De acuerdo con la Organización de las Naciones Unidas para la Alimentación y la Agricultura 1,300 millones de toneladas anuales se pierden o desperdician a escala mundial. Generalmente se desperdicia el 50% de frutas y verduras, 30% de cereales, 60% de tubérculos, y 20% de alimentos de origen animal. FAO (2012). Se estima que el 6% de las pérdidas mundiales de alimentos se dan en América Latina y el Caribe. En América Latina se pierde el 15% de los alimentos que se producen cada año, es decir 80 millones de toneladas, lo que significa que cada habitante desperdicia 220 kg de alimentos al año.

Con respecto a México el desperdicio es de más de 10 millones de toneladas de alimentos anuales, lo cual representa el 37% de la producción agropecuaria en el país. Las pérdidas de alimentos en México tienen lugar en las etapas de procesamiento 6%, mercado 17%, manejo y almacenamiento 22%, producción 28% al igual que en hogares.

1.2. Objetivo

Desarrollar una aplicación dirigida a las amas de casa que le permita llevar un control de entradas y salidas de sus alimentos.

1.3. App Inventor 2

App Inventor es una aplicación originalmente desarrollada por Google y mantenida ahora por el Instituto de Tecnología de Massachusetts. Permite que cualquier persona, incluyendo las no familiarizadas con la programación y SDK de Android, pueda crear aplicaciones de Software para Android. Utiliza una interfaz gráfica, muy similar al *Scratch* y el *StarLogo*, que permite a los usuarios arrastrar y soltar objetos visuales para crear una aplicación que puede ejecutarse en el sistema Android. Google puso fin al desarrollo el 31 de diciembre de 2011 cediéndole el código al MIT, quien lo ha puesto a disposición de todos. Se trata de una utilidad Web desarrollada por Google que permite realizar aplicaciones para Android sin escribir código, todo de forma visual e intuitiva (uniendo piezas de un puzle). Utiliza una interfaz gráfica que permite a los usuarios arrastrar y soltar objetos visuales para crear una aplicación que puede ejecutarse en el sistema Android que funciona en muchos dispositivos móviles. Todo ello sin usar ni una sola línea de código, de forma intuitiva y gráfica. Molina, (2006).

En la creación de App Inventor para Android, Google se basó en la investigación de la informática educativa, y el trabajo realizado en entornos de desarrollo en línea. El editor de bloques utiliza la biblioteca *Open Blocks* de Java para la creación de lenguajes de programación visuales. *Open Blocks* está distribuida por el *Massachusetts Institute of Technology Program`s Scheller* para formación de profesorado y deriva de la investigación de la tesis de Ricarose Roque. El profesor Eric Klopfer y Daniel Wendel del Programa Scheller apoyaron la distribución de bloques abiertos bajo la licencia MIT. La programación de bloques abiertos y visuales está estrechamente relacionada con el *StarLogo*, un proyecto de Klopfer, y Scratch. Estos proyectos están formados por teorías del aprendizaje construccionista, que hace hincapié en que la programación puede ser un vehículo para conseguir ideas de gran alcance a través del aprendizaje activo. Como tal, es parte de un movimiento continuo en las computadoras y la educación que se inició con el trabajo de Seymour Papert y el Grupo de *StarLogo* del MIT en 1960, y también manifestado en el trabajo de Mitchel Resnick, Lego Mindstorms y *StarLogo*. El equipo de *App Inventor* fue dirigido por Hal Abelson y Mark Friedman.

2. Diseño metodológico

Esta investigación se realizó en dos fases:

- La estrategia metodológica (ver fig. 1), que implicó: el contexto, el marco teórico, el problema y los tipos de investigación, con base en ello se estableció el diseño y la segunda fase.
- El trabajo empírico, que consistió en la obtención, análisis y presentación de la aplicación. En esta etapa se incluye la metodología de desarrollo del software, así como el desarrollo de la interfaz.

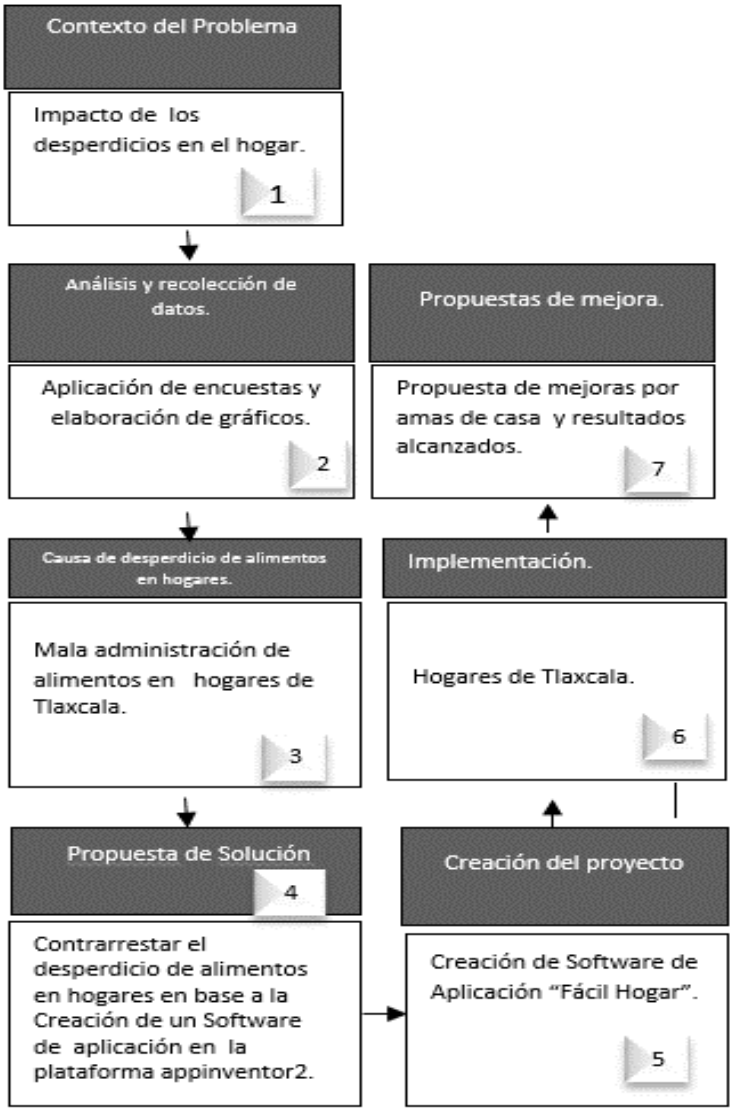


Fig. 1. Estratégia metodológica.

2.1. Metodología de desarrollo de software

La figura 2, muestra la metodología empleada para el desarrollo del software "fácil hogar" misma que fue selecciona luego de revisar las propuestas en JISBD (2003), Yusef (2004), Sommerville (2005) y Schwaber (2013), a fin de generar un producto de manera rápida y funcional. A su vez, tomando en consideración la investigación de Catalán V., se diseñó la interfaz de usuario en una primera etapa de tal forma que fuese amigable con el usuario, en este caso la ama de casa.

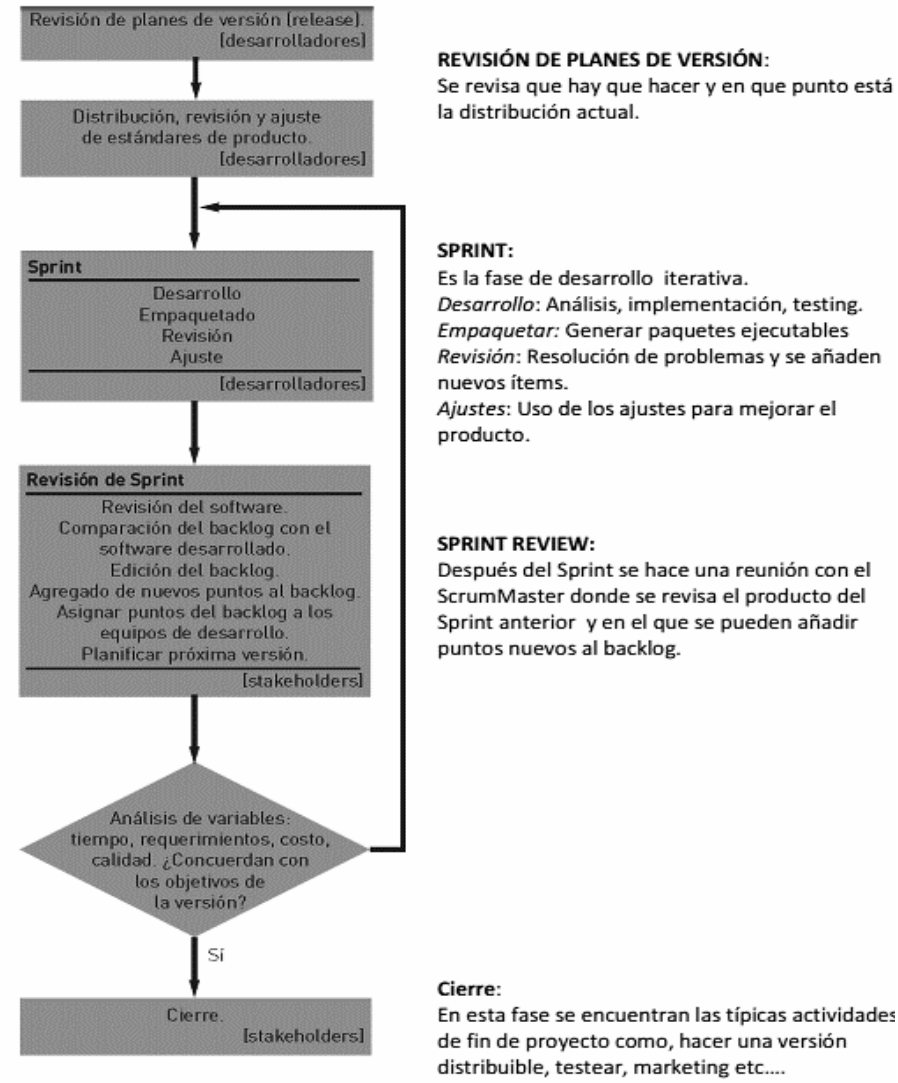


Fig. 2. Metodología Scrum para desarrollo de Software.

3. Desarrollo

3.1. Contexto del Problema

En 2007, se utilizaron casi 1.400 millones de hectáreas de tierras para producir alimentos que no se consumieron. Esto representa una superficie más grande que Canadá y la India juntos. El despilfarro de alimentos contribuye a la expansión agrícola hacia zonas silvestres lo que se traduce en pérdidas de especies, incluidos mamíferos,

aves, peces y anfibios. Los alimentos se pierden o desperdician en toda la Cadena de Suministro, en México existe una mayor pérdida de alimentos en el sector de consumo es decir en los hogares, con una de pérdida del 28%. Las causas de desperdicio de alimentos en los hogares se basan principalmente en aspectos relacionados con las fechas de consumo. McMahon (2011).

Esta misma problemática se ve reflejada en los hogares de la entidad de Tlaxcala que se han visto afectados por la pérdida de alimentos debido a la fecha de caducidad teniendo un impacto negativo en la economía de los hogares. La investigación se aplica en la ciudad de Apizaco Tlaxcala debido a que es una zona urbana y las amas de casa están familiarizadas con el uso de la tecnología.

3.2. Análisis y recolección de datos

Con el propósito de obtener la información sobre el desarrollo de la aplicación "fácil hogar" se llevó a cabo una encuesta de opinión. Tomando una muestra con un nivel de confianza de 95% y se aplicó la encuesta a 300 amas de casa. La encuesta consistió en una entrevista directa y personal, en las cuales se les cuestionaron los aspectos diversos sobre el desperdicio de alimentos como:

- ¿En qué basa el orden de consumo de sus alimentos?
- ¿Hace uso de las tecnologías (Tablet, Celular, Internet)?

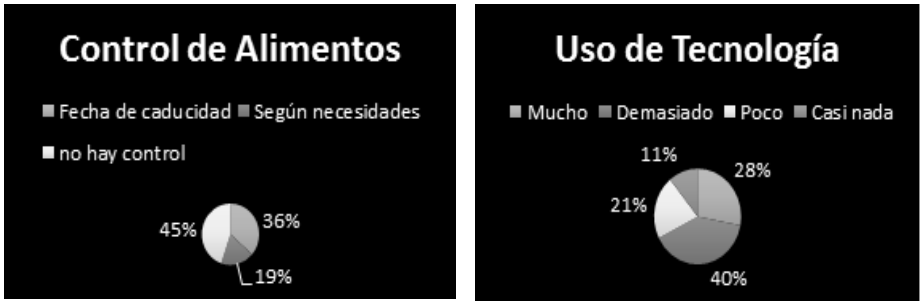


Fig. 3. Control de alimentos y uso de tecnología (creación propia).

Como resultado de la primera pregunta se obtuvo que el 45% de los encuestados no tiene un control sobre las entradas y salidas de sus productos lo que ocasiona una elevada cantidad de desperdicio y un incremento considerable en sus costos, mientras que el 36% no toma en cuenta las fechas de caducidad de los producto, y por último con un 19% las compras las realiza de acuerdo a sus necesidades. En la figura 3, se muestra los resultados obtenidos de la encuesta realizada a las amas de casa en cuanto al control de los alimentos y el uso de la tecnología. Respecto los resultados obtenidos con respecto a las amas de casa que utilizan tecnología (Tablet, celular, internet), el 40% de las amas de casa están en contacto directo con la tecnología, el 28% muestra un uso frecuente con celular o tablet. Estos indicadores nos muestran que la aplicación a desarrollar será de uso frecuente para la administración de los alimentos.

3.3. Causa de desperdicio de alimentos en los hogares de Tlaxcala

En base a los resultados obtenidos anteriormente se encontró que la causa principal de desperdicio en los hogares de Apizaco es que el ama de casa no tiene una correcta administración del consumo de sus alimentos, debido a que ignora la fecha de caducidad de los productos, consumiendo en primer lugar los que tienen una fecha de caducidad más alejada y dejando rezagar los próximos a caducar.

Después del análisis de la situación e identificación de las principales causas de desperdicio de alimentos se procede a la etapa 4, la propuesta de solución en la cual se propone diseñar y construir un Software de aplicación para los celulares o tabletas llamado "fácil hogar", el cual consiste en un conjunto de formularios que permite al ama de casa visualizar de manera rápida y fácil los alimentos que están más próximos a caducar los cuales deben ser consumidos en primer lugar, así como también le permite llevar un efectivo control de existencias, todo esto con la finalidad de contrarrestar el desperdicio de alimentos y optimizar los costos.

3.4. Creación del software

La creación del software de Aplicación se realizó en la plataforma de *App Inventor2* en 5 fases, se describen como sigue:

1. Formulario de Inicio. Permite a la ama de casa ingresar a la aplicación por medio de reconocimiento de voz, utilizando el componente *Speech Recognizer* el cual permite reconocer patrones de voz y llevarlas a texto. En la figura 4 se muestra el formulario y los bloques utilizados para su programación.

Fig. 4. Formulario de ingreso (creación propia).

2. Formulario de Menú. Le permite a la ama de casa ingresar a los diferentes formularios de la aplicación según sus necesidades, en este formulario se utiliza el componente *Open Another Screen*, este componente ayuda a la programación de los botones para crear un vínculo de un formulario a otro. En la gura 5 se muestra el formulario y sus bloques de programación correspondientes.

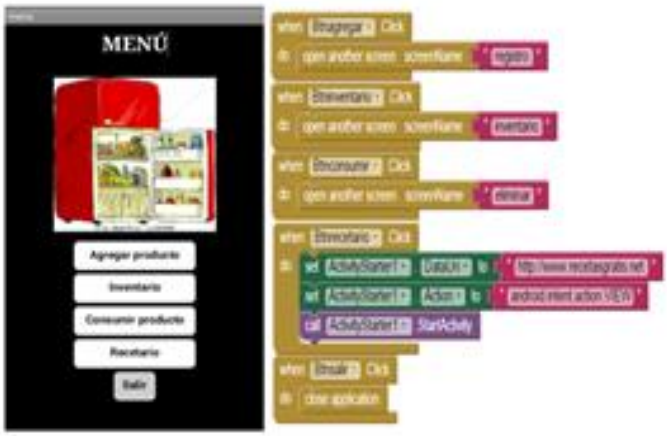


Fig. 5. Formulario del menú (creación propia).

3. Formulario de registro de producto. En este formulario la Ama de Casa podrá registrar sus productos del refrigerador o la alacena, para el funcionamiento correcto del registro se necesita descargar un lector de barras llamado *Barcoder Scaner* ya que es indispensable registrar este campo para que sirva como elemento de búsqueda en otros formularios, también se utiliza el componente *Date Picker* para ajustar la fecha actual, los datos que se ocupan son nombre del producto, fecha de caducidad, código de barras. En la gura 6 se muestra el formulario de registro de productos.

Fig. 6. Formulario de productos (creación propia).

4. Formulario de consumo. En este formulario el ama de casa podrá eliminar de su base de datos los productos que se han consumido, para programar esta aplicación se utilizaron los comandos llamados *Delete* que permiten eliminar de manera permanente los productos de la base de datos. Para poder elegir los productos almacenados en la memoria interna del dispositivo se requiere que el producto sea buscado por medio del

código de barras ya que es un dato que no se repite en ningún producto. En la figura 7 se muestra el formulario y los bloques programados para el correcto funcionamiento del formulario.

Fig. 7. Formulario de consumo (creación propia).

5. Formulario de Inventario. En este formulario el ama de casa puede visualizar la existencia de sus productos, los productos son almacenados en la memoria interna del dispositivo con la ayuda de la base de datos *Tynni* este elemento permite almacenar los datos en la memoria interna del dispositivo, este formulario contiene el elemento *Show lterbar* uno de los más importantes de la aplicación, en este se pueden filtrar los productos por nombre o por fecha de caducidad para conocer cuáles son los productos que se deben de consumir en primer lugar. En la Figura 8 se muestra el formulario y los bloques de programación delInventario.

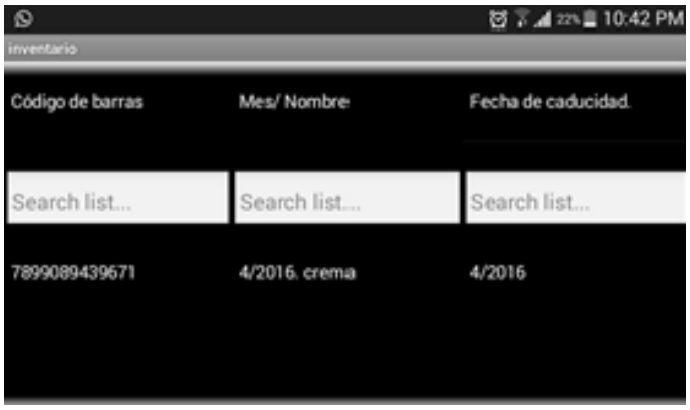


Fig. 8. Formulario de inventario (creación propia).

Debido a las características de la plataforma utilizada para la Creación de este Software de Aplicación, solo podrá funcionar si se cuenta con una conexión Wi-Fi.

4. Resultados obtenidos

Para comprobar la funcionalidad del Software de Aplicación "fácil hogar, se procedió a la implementación" en 10 hogares de la ciudad de Apizaco Tlaxcala del 1 al 30 de Abril del 2016, teniendo una participación favorable de las Amas de casa, ya que estaban muy interesadas por aprender a usar la Aplicación obteniéndose los siguientes resultados.

Fig. 9. Comparación de costos antes y después de la implementación.

En las gráficas de la figura 9, puede observarse la comparación de los costos de las amas de casa en alimentos antes de la implementación de la aplicación. El ama de casa gastaba entre $2000 y $3000 pesos y después del uso de la Aplicación gastó de $800 a $1000 pesos teniendo como resultado un ahorro de $1500 pesos por hogar.

5. Conclusiones

El desperdicio de alimentos es mayor aún en un hogar, debido a que los alimentos que se tienen almacenados se ha dejado que se estropeen o caduquen por negligencia, se estima que el valor de los alimentos que se pierden o desperdician cada año en todo el mundo es de un billón de USD. Ahora bien las causas de desperdicio de alimentos en el hogar están principalmente relacionadas con el comportamiento del consumidor como planificar inadecuadamente las compras y no consumir los alimentos antes de su fecha de caducidad también conlleva un desperdicio de alimentos evitable. Es por ello la importancia de comprender primeras entradas, primeras salidas, considerando obtener reducción de costos, y desperdicios ya que las pérdidas de alimentos representan una pérdida del valor económico para los actores de las cadenas de producción y suministro de alimentos. En ultimátum recae en primeras entradas, primeras salidas, técnica de gestión que trata de distribuir los alimentos por la cadena de suministro seleccionando primero los que caduquen antes (First Expires) y, al igual de caducidad más antigua (First Out). Así la caducidad se convierte en el eje sobre el que gira todo el funcionamiento de un almacén. En este lapso de dominar el control de caducidades de alimentos con primeras entradas y primeras salidas, se incorporan diferentes formas para evitar desperdicios y reducir costos, como el apoyo de las

tecnologías con uso de aplicaciones, en éste caso. Se ha determinado que implementando nuevas tecnologías al uso cotidiano de un hogar, puede apoyar en disminuir los índices de desperdicios y mejorar los niveles de alimentación. El ama de casa puede realizar sus labores domésticas de forma interactiva y controlada, con el Software de aplicación tendrá a su alcance los productos y alimentos comprados desde un supermercado y así facilitar su consumo. Este proyecto permitió estimular el enfoque educativo y alimenticio ya que su desarrollo se basó sobre la disminución de desperdicio en un hogar con un funcionamiento exitoso debido a que se ha disminuido considerablemente el desperdicio de alimentos además de que se observó una reducción en los gastos alimentarios de la muestra.

Referencias

1. INEGI: Estadísticas históricas de México. 8, Agropecuario, aprovechamiento forestal y pesca, México: INEGI (2009)
2. Molina, J., Córdova, L.: App Inventor Mit 2 (2006)
3. Censo Agropecuario: Panorama Agropecuario en México: Censo Agropecuario. Instituto Nacional de Estadística y Geografía, México, INEGI (2007)
4. McMahon, M., Valdés, A., Cahill, C., Jankowska, A.: Análisis del Extensionismo Agrícola en México, México, SAGARPA (2011)
5. FAO: Pérdidas y Desperdicio de Alimentos en el Mundo, Alcance, Causas y Prevención. Roma, FAO (2012)
6. Wyman, O.: Reducir el desperdicio de los alimentos, ¿Cómo pueden las empresas de distribución ayudar? Estados Unidos de Norte América (2014)
7. FAO: Iniciativa mundial sobre la reducción de la pérdida y el desperdicio de alimentos. Roma, FAO (2014)
8. HLPE: Las pérdidas y el desperdicio de alimentos en el contexto de sistemas alimentarios sostenibles. Un informe del Grupo de alto nivel de expertos en seguridad alimentaria y nutrición del Comité de Seguridad Alimentaria Mundial, Roma (2014)
9. Taller Metodologías Agiles en el Desarrollo de Software: Taller realizado en el marco de las VIII Jornadas de Ingeniería del Software y Bases de Datos, Universidad Politécnica de Valencia, JISBD (2003)
10. Hassan, Y., Martín, F. J., Iazza, G.: Diseño Web Centrado en el Usuario: Usabilidad y Arquitectura de la Información. Hipertext.net, No. 2, http://www.hipertext.net (2004)
11. Sommerville, I.: Ingeniería del Software. Séptima edición, Pearson (2005)
12. Schwaber, K., Sutherland, J.: La Guía de Scrum, Las Reglas del Juego (2013)
13. Catalán, V. M.: Metodología de evaluación de interfaces gráficas de usuario. http://eprints.rclis.org/6732/1/Metodologias_de_evaluaci%C3%B3n_de_interfaces_graficas_de_usuario.pdf

ISSN 1870-4069

Diseño y desarrollo de una aplicación móvil accesible de navegación individual y localización para personas de la tercera edad con discapacidad visual

Etelvina Archundia, Carmen Cerón, Patricia Cervantes, Fernando Rodríguez

Benemérita Universidad Autónoma de Puebla,
Facultad de Ciencias de la Computación,
México

{etelvina, mceron}@cs.buap.mx,
{cervantes.patty, ferrgzt}@gmail.com

Resumen. El propósito del artículo se centra en el *Diseño y desarrollo de una aplicación móvil para apoyar la navegación individual y localización de personas de la tercera edad con discapacidad visual.* Para el desarrollo de la aplicación móvil, se utilizó la metodología *Extreme Programación, el Diseño Universal, y las tecnologías integradas de WebService, PHP, MYSQL, Java y Apache.* La aplicación permite la ubicación del sujeto y compartirla entre distintos usuarios, con la finalidad de brindar seguridad y facilitar una mayor inclusión del uso de la tecnología a esta población. Finalmente se presentan los resultados obtenidos de una prueba piloto mediante la experiencia de los usuarios en distintos escenarios.

Palabras clave: Discapacidad visual**,** usabilidad, apps, adulto mayor.

Design and Development of an Accessible Mobile App of Individual Navigation and Localization for Elder People with Visual Disability

Abstract: The purpose of this article is focused on the design and development of an accessible mobile app of individual navigation and location for elder people with visual disability. To develop the mobile application, it was used as methodology *Extreme Programming, el Universal Design, integrated technologies of WebService, PHP, MYSQL, Java and Apache.* The app allows to know the man's location and share it within many users providing security and facilitate a better inclusion of the use of technology to this kind of population. Finally the article shows results obtained from a pilot test through the user's experience in different scenarios.

Keywords: Visual disability, usability, apps, elder people.

1. Introducción

Las tecnologías de la información y la comunicación (TIC) proporcionan, en principio, los mismos beneficios a las personas mayores que al resto de la población: acceso a la comunicación, información, ocio, servicios que desde internet facilitan la vida diaria. Sin embargo se requieren aplicaciones accesibles para las personas mayores que permitan manejarlas con la menor carga cognitiva e interfaces fáciles de manejar. Además se han identificado en las personas de la *tercera edad* alguna discapacidad auditiva o visual, disminuyendo la agudeza visual, auditiva y la destreza manual, esto conlleva a ser necesario desarrollar herramientas o artefactos tecnológicos para apoyarlos en su entorno de trabajo, familiar y social [1].

El actual envejecimiento de la población es un acontecimiento demográfico que nunca antes se había vivido en la historia de la humanidad dado el descenso de la natalidad y la prolongación de la vida media de las personas. Los índices de envejecimiento siguen creciendo desde hace años de forma desmesurada lo que conlleva a importantes repercusiones a nivel social y sanitario [2]. Por otro lado, vivimos en una sociedad donde el volumen de nuevos conocimientos e información crece de forma acelerada y en progresión geométrica en todos los sectores de la vida social. A este hecho contribuye el desarrollo alcanzado por las denominadas Tecnologías de la Información y la Comunicación (TIC), que juegan un papel muy importante tanto en la creación como en la difusión de la información y que pueden ayudar de forma importante a las personas mayores en su formación formal o informal a lo largo de la vida [3]. De ahí la importancia de conseguir competencias para la alfabetización digital no solamente técnicas sino también pedagógicas dado el enorme potencial de las herramientas digitales [4].

A finales del 2013, la Organización de Naciones Unidas (ONU) informó que para el año 2050 estiman más de dos mil millones de adultos de la *tercera edad* usarán Internet, lo cual representa un aumento del 300% [5]. Lo cual implica que esta población hará uso de distintas aplicaciones y dispositivos tecnológicos con mayor frecuencia en las actividades cotidianas que desempeñe. El uso de internet, redes sociales o desde otras aplicaciones permite compartir información entre usuarios, siendo la ubicación o localización una forma de navegar para compartirla en las aplicaciones. Esto requiere aplicaciones más seguras para las personas mayores, ya que son más vulnerables a extraviarse o requerir apoyo para ubicar algún lugar o sitio a donde se desplacen.

Con base a lo anterior, el presente trabajo tiene como propósito el diseño y desarrollo de una Aplicación Móvil para dispositivos con sistema Android con la finalidad de servir como una herramienta de navegación personal para compartir la ubicación de manera rápida y sencilla de los usuarios de la *tercera edad* con las personas deseadas mediante el uso de un código, almacenado en una base de datos, lo cual brindará mayor seguridad para las personas de la *tercera edad.* Por otra parte el diseño permite la configuración de interfaces accesibles móviles para las personas de la *tercera edad* con discapacidad auditiva y/o visual.

La aplicación permitirá ubicar al usuario de manera exacta, generar un código único y almacenarlo en la base de datos. Así como la obtención de la información y mostrar la ubicación real del usuario al utilizar Web Services.

El artículo se organiza de la siguiente manera: en la sección 2 presenta la fundamentación teórica y el estado del arte de este trabajo; en la sección 3 se realiza el análisis y diseño de la aplicación móvil; en la sección 4 se presenta la propuesta de la aplicación móvil y la prueba piloto. Finalmente, se presentan las conclusiones y el trabajo futuro en la sección 5.

2. Marco teórico

En esta sección revisaremos los tópicos de las personas de la *tercera edad*, discapacidad visual y la tecnología. Así como los tipos de aplicaciones móviles, el diseño centrado en el usuario y la experiencia del usuario, siendo relevantes en nuestra propuesta de trabajo.

2.1. Personas de la tercera edad, discapacidad visual y tecnología

En general, la tecnología ha tenido que proponer diferentes adaptaciones para poder ser incluyente y accesible a las personas con discapacidades y de la t*ercera edad*. En ese sentido ha surgido la tecnología de apoyo conocida también como "tecnología de adaptación o rehabilitación". Según Alcantud la define como: "Todos aquellos aparatos, utensilios, herramientas, programas de ordenador o servicios de apoyo que tienen como objetivo incrementar las capacidades de las personas que, por cualquier circunstancia, no alcanzan los niveles medios de ejecución que por su edad y sexo le corresponderían en relación con la población normal" [6]. En la convención de las Naciones Unidas sobre los Derechos de las Personas con Discapacidad, aprobada en 2006 [7], existe la necesidad que se diseñen aplicaciones bajo un diseño universal para todos, pero que no excluye las ayudas técnicas para grupos particulares de personas con distintas discapacidades, lo cual requieren que se adapten a las capacidades específicas, de acuerdo a las normas de accesibilidad, tanto para contenidos, servicios y dispositivos.

2.2. Aplicaciones móviles accesibles

Los dispositivos móviles, incluyendo los teléfonos inteligentes y las tabletas, permiten a los usuarios de la t*ercera edad* acceder de forma más sencilla a los servicios de internet a través de las redes móviles e inalámbricas. En el 2012, el mercado mundial móvil se conformó en más de 6500 millones de usuarios, la mayoría de los cuales proceden de países en desarrollo [8] generando la necesidad de desarrollar diversas aplicaciones móviles para distintas generaciones.

Según Gil [9], las aplicaciones móviles conocidas como "Apps", ya se han establecido principios básicos para el diseño de "Apps Accesibles" de acuerdo a la norma "UNE 139802:2009-Requisitos de accesibilidad del software. (ISO 9241-

171:2008)", los cuales han sido ya adaptados a las necesidades de los dispositivos móviles. Las aplicaciones móviles accesibles han sido definidas como "una aplicación es accesible cuando cualquier usuario, independientemente de su diversidad funcional, puede utilizarla en su dispositivo móvil satisfactoriamente con su sistema de acceso habitual" [9].

Una aplicación móvil se identifica en la actualidad porque puede funcionar en distintos dispositivos móviles como característica principal, sin embargo, dentro de las aplicaciones móviles existen tres tipos [9]:

- Aplicaciones nativas, son aquellas que se desarrollan bajo un lenguaje y un entorno de desarrollo en específico para cada sistema operativo.
- Aplicaciones web, estas son desarrolladas usando leguajes para el desarrollo web como lo son HTML, CSS y JavaScript.
- Aplicaciones Hibridas, como su nombre lo indica tienen un poco de cada tipo de las aplicaciones anteriores.

Las aplicaciones móviles las podemos encontrar de forma gratuita o con un costo en los diferentes sitios como: "AppStore", "GooglePlay" y "Windows PhoneStore", quienes ofrecen servicios de compras en línea, educación, finanzas, navegación, etc., siendo unas de las aplicaciones más utilizadas la geolocalización, la cual ofrece servicios para facilitar las tareas como: mostrar el tráfico, optimización de rutas, lugares de interés social, recordatorios al pasar cierta zona, ver los lugares más visitados, buscar direcciones y otros servicios. Las aplicaciones más usadas en este contexto son:

- Google Maps, es una aplicación móvil de mapeo desarrollado por Google para los sistemas operativos Android e iOS, utiliza la API de Google Maps para su información. Los mapas e información no están incluidos en los mapas de Google instaladas para los dispositivos, se requiere una conexión a Internet para hacer uso de ellos y permite al usuario, descargar la ruta de un punto de referencia a otro, con un área de 10 millas cuadradas (26 km2) alrededor de cualquier punto [10].

- Waze, es la aplicación de tráfico y navegación basada en la comunidad más grande del mundo, dando información de tráfico y ruta en tiempo real por sus usuarios [11].

- Foursquare, consiste en una red social, en la que los usuarios realizan 'check-ins' en los lugares que visitan, comparten recomendaciones y opiniones con sus contactos. Para poder hacerlo de manera efectiva hace uso del GPS del dispositivo [12].

2.3. Sistemas de posicionamiento

Los sistemas de posicionamiento, se refieren al método por el cual se puede determinar la posición de un dispositivo, y utilizando servicios adicionales se puede responder preguntas como ¿Dónde estoy?, ¿Qué hay cerca?, ¿Cómo llego allá?, entre otras.

La ubicación puede ser expresada como una descripción textual o en términos espaciales [11]. Una ubicación expresada como un texto es por lo general la dirección

de una calle, la ciudad, comuna, código postal, etc. Una ubicación espacial puede ser expresada en términos de coordenadas geográficas usando latitud-longitud-altitud (esta última opcional). La latitud se expresa en grados de 0-90 al norte o sur del ecuador, la longitud en grados de 0-180 al este u oeste del meridiano de Greenwich, y la altitud en metros sobre el nivel del mar. Existen varios métodos [12] para determinar la ubicación de un dispositivo, los cuales varían en las tecnologías empleadas y en la precisión con que entregan el resultado. Siendo las más utilizadas las redes telefónicas, celulares, de área local, Wi-Fi, Bluetooth y GPS entre otros.

- Con respecto al *método Cell-ID*, es considerado de bajo costo que utiliza la red GSM para determinar la ubicación de un teléfono celular. El identificador de la antena o célula (*Cell ID)* a la que está conectado el dispositivo es usado para aproximar la ubicación del usuario. La precisión de este método depende del tamaño de la célula: unos 500 metros de exactitud dentro de una ciudad y unos 10 kilómetros en zonas rurales.
- Por otra parte, el *método Bluetooth*, es una red de corto alcance, este provee un posicionamiento de buena precisión, con máximo de error de unos 10 metros, además de poder entregar posicionamiento vertical. Es muy adecuado para obtener la ubicación dentro de edificios o zonas urbanas pequeñas.

2.4. El enfoque de la experiencia del usuario

Para Nielsen y Norman, afirman de la necesidad de distinguir experiencia del usuario y usabilidad, considerando que la usabilidad es un atributo de calidad de una interfaz de usuario asociada a la facilidad de aprender a usar un sistema, su uso eficiente y placentero, mientras que la experiencia del usuario es un concepto más amplio que involucra el análisis de la experiencia de interacción más allá de la relación entre usuario y producto [13].

3. Metodología de desarrollo de la aplicación móvil

En esta sección se presenta el análisis, diseño y arquitectura de la aplicación, bajo el enfoque de la metodología ágil y el diseño centrado en el usuario.

3.1. Metodología del diseño centrado en el usuario y desarrollo de software ágil

El Diseño Centrado en el Usuario (DCU) es aquel que desde los inicios del desarrollo del producto, hace participar al usuario y lo involucra durante todo el proceso. En este trabajo, se usará una propuesta de DCU Ágil. Esta metodología combina el proceso típico del Diseño Centrado en el Usuario y Desarrollo de Software Ágil (Programación extrema XP, como se muestra en la Figura 1.

En la fase de investigación, necesidades de usuarios y contextos de uso: se ha usado la técnica de investigación contextual (contextual Enquiry) que permite

observar la motivación del usuario mientras interactúa con un sistema similar al que se va a diseñar. Con respecto a la programación Extrema [14], se aplicó y con la fase de exploración, se determinaron los requerimientos por medio de la creación de *Historias de usuarios* del grupo focal: "Adultos mayores discapacitados" y se integraron los casos de uso en UML para la aplicación.

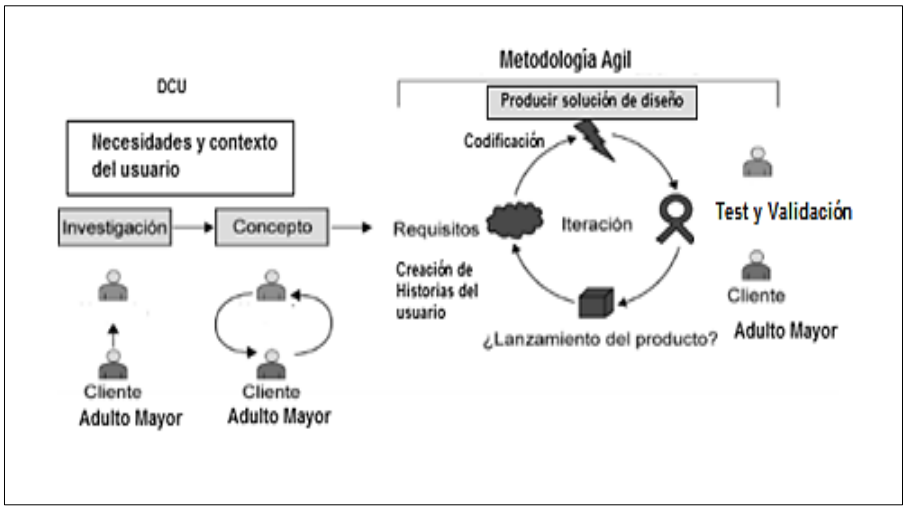


Fig. 1. Metodología de desarrollo de la aplicación móvil.

En la Figura 2, se muestra el usuario "*Adulto Mayor*" con las actividades a desempeñar y la interacción que tiene con la base de datos del dispositivo y el WebService.

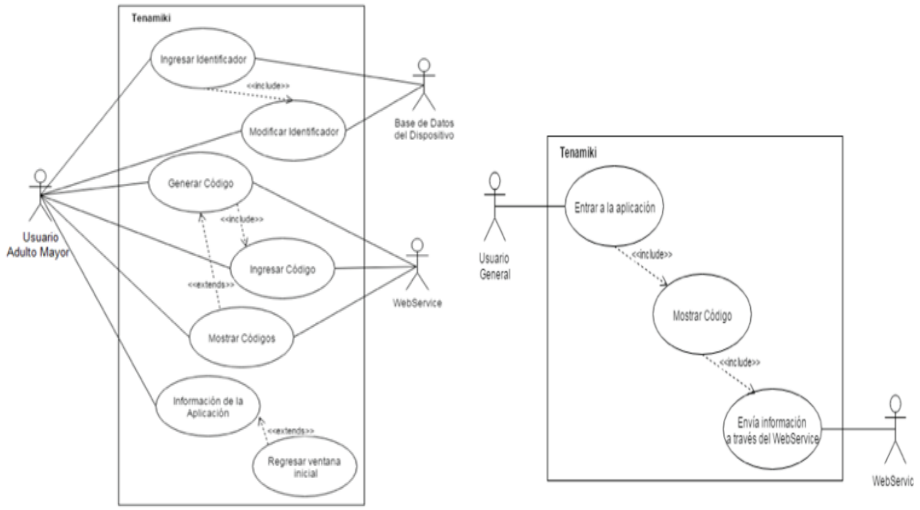


Fig. 2. Casos de uso del sistema.

3.2. Arquitectura de la aplicación móvil

La arquitectura de la aplicación consta de dos partes: la primera parte es la aplicación para dispositivos Android, la cual se conecta al WebService [15], que a su vez se comunica con la base de datos que almacena e intercambia información de la aplicación. Por otra parte la visión de un módulo Web agregado al servidor web con *Apache* para el portal de aplicaciones de apoyo a adultos mayores *de la tercera edad Accesibles (AAMA)*, como se muestra en la Figura 3.

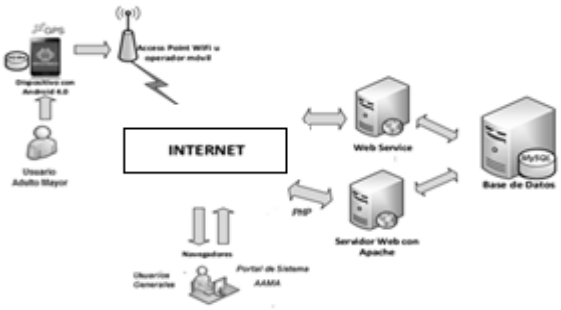


Fig. 3. Esquema de la Arquitectura de la aplicación móvil.

3.3. Producir prototipo de la aplicación móvil

El Diseño centrado en el usuario se adoptó para las características propias del usuario de *Adulto Mayor* con alguna discapacidad visual. Para lo cual se realizó una serie de entrevistas y encuestas con el grupo focal, finalmente se plantearán los prototipos de la aplicación, los cuales se realizaron con el *Software Balsamic*, como se muestra en la Figura 4.

Fig. 4. Esquema de prototipos del la interfaz de la aplicación móvil.

Los resultados obtenidos fue que la mayoría de los *adultos mayores de la tercera edad* ya poseen un teléfono inteligente, siendo el sistema operativo que más predomina es *Android* y de las herramientas que han muy poco utilizado es *Google Maps y Waze*, pero aseguran que requieren no solo compartir su localización, sino tener certeza de que hayan recibido la localización sus familiares o conocidos y saber cuántos usuarios han consultado la ubicación que han compartido y poder llevar un registro de las personas contactadas por este medio y los datos de la localización de los usuarios.

Por lo cual se planteó las características principales con las que debe contar la aplicación:

- Interface clara e intuitiva: Puesto que lo han de utilizar personas con muy pocas habilidades tecnológicas, se requiere de una interface gráfica muy clara e intuitiva, con pocos botones y que los que haya sean grandes. Además de incluir el modo auditivo de lectura de textos y sonidos.
- Pocas opciones hasta llegar al resultado: Se quiere que la aplicación sea ágil, así que se necesita pocas pantallas y decisiones, para llegar al resultado esperado.
- **Práctica**: La aplicación debe resultar lo más práctica posible para que el usuario la encuentre de gran utilidad.

4. Desarrollo y pruebas del prototipo de la aplicación móvil

En esta sección se muestran los resultados del prototipo de la implementación de la aplicación móvil así como las pruebas de funcionalidad junto con las pruebas de usabilidad obtenidas del grupo focal de usuarios. Las tecnologías utilizadas PHP, *MySQL, Webservices, Java, Android Studio 15, Servidor Web: Apache 2.4.10, Emulador Android: Genymotio 2.5.3.* Las interfaces finales para la aplicación móvil TENAMIKI (significa en Náhuatl "Encontrase") como se muestran en las Figura 5.

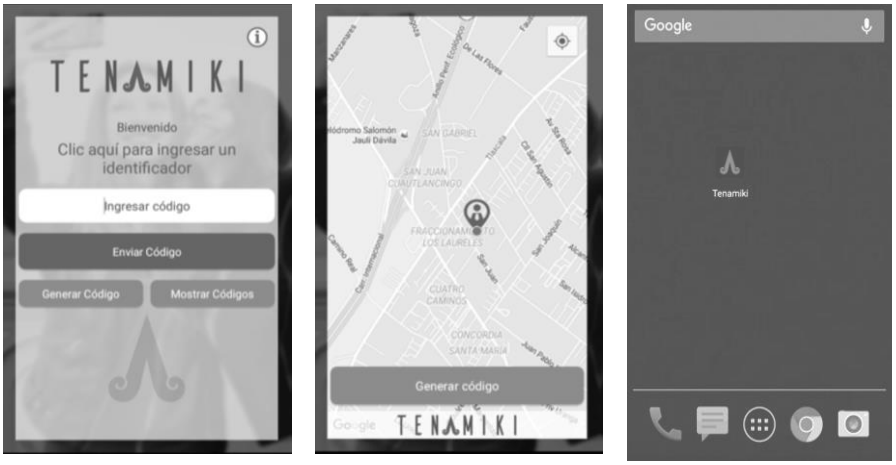


Fig. 5. Pantalla de inicio con elementos visuales para la tercera edad.

4.1. Prueba Piloto y de Usabilidad

La aplicación móvil se le aplicó una prueba piloto un grupo de 15 usuarios adultos mayores de la *tercera edad*, de los cuales el 80% tiene una disminución visual. Para la evaluación de la aplicación, se realizaron pruebas de funcionalidad y usabilidad. Con respecto a las pruebas de Usabilidad presentaron tres posibles escenarios posibles para los usuarios finales, a continuación se describen:

- Situación 1: Al usuario se le entregó la aplicación con la WiFi, GPS y 3G apagados, además se les dio una breve explicación del uso de la aplicación.
- Situación 2: Al usuario se le entregó la aplicación con la WiFi, GPS y 3G prendidos y se le explico el uso de la aplicación.
- Situación 3: Al usuario se le entregó la aplicación con la WiFi, GPS y 3G apagados y no se explicó el uso de la aplicación

Para cada una de las situaciones los usuarios deberán cumplir ciertas tareas para así comprobar el funcionamiento de la aplicación.

- Tarea 1: Ingresar un código generado.
- Tarea 2: Encontrar dónde se muestran los códigos que se han generado.
- Tarea 3: Generar un código en una localización diferente a la que se encuentra actualmente.
- Tarea 4: Cambiar el identificador.
- Tarea 5: Compartir el código.

Después de realizar las pruebas a 15 usuarios 5 por cada situación, se muestran los resultados obtenidos en la Tabla 1, donde se observa el porcentaje de usuarios que pudieron completar la tarea.

Tabla 1. Resultados Prueba de Usabilidad usando Escenarios.

	Escenarios de Test		
Tareas	**Situación 1**	**Situación 2**	**Situación 3**
Tarea no.1	90%	95%	60%
Tarea no.2	100%	100%	100%
Tarea no.3	100%	100%	100%
Tarea no.4	100%	100%	100%
Tarea no.5	80%	100%	80%
Promedio	94%	99%	88%

Lo cual refleja que los usuarios con una breve explicación de la situación 1 su desempeño fue del 94% del cumplimiento de las tareas mientras que los usuarios de la situación 2 al 99% realizaron las tareas casi en su totalidad y para la situación 3 lograron las tareas en un 88%, esto implica que se presenta una interfaz intuitiva y agradable. Finalmente se aplicó una encuesta de satisfacción de valoración del software [16, 17] la evalúa siete criterios: navegación, interactividad, inmersión, usabilidad, creatividad, efectividad y calidad, con una escala de 1 a 5, cuyo promedio

obtenido fue de 4.7 (94.37%), a continuación se presentan los resultados en la Figura 6.

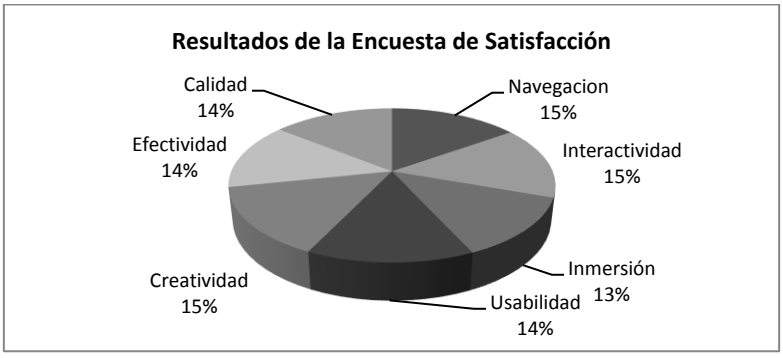


Fig. 6. Resultados de la encuesta de satisfacción de la aplicación móvil.

5. Conclusiones y trabajo a futuro

El diseño y desarrollo de una aplicación móvil para dispositivos con sistema *Android* que permita compartir su ubicación de manera rápida y sencilla a los usuarios de la *tercera edad* con discapacidad visual, mediante la generación de un código, se implementó alcanzando un porcentaje mayor al 90% de satisfacción. La Metodología del *Diseño Centrado en el Usuario y Desarrollo de Software Ágil*, permitió el identificar las características de los adultos mayores de la *tercera edad* con discapacidad visual en el contexto de la investigación.

Los adultos de la *tercera edad* se consideran la sociedad cognitiva a la cual se requiere de estudios para integrarlos al uso de las aplicaciones móviles accesible, a través de interfaces amigables para la discapacidad visual y evitar la marginación de personas todavía productivas en su vida, por ello se pretende, brindar un aporte, más allá de las diferencias de capacidades, y se beneficien con las herramientas informáticas como un medio de inserción en un mundo que ofrece cada vez mayores posibilidades de integración en la sociedad.

Las investigaciones que se continúan desarrollando se integrarán en una aplicación para las personas de la *tercera edad con alguna discapacidad*: auditiva, cognitiva y por supuesto visual; puesto que se requiere contribuir a través de las *Tecnologías de la Información y Comunicación* para la contextualización social y productividad con calidad de vida.

Referencias

1. Muñoz, C.: Bienestar Subjetivo y Actividad Social con Sentido Histórico en Adultos Mayores. Revista Hacia la Promoción de la Salud, Disponible en: http://google.redalyc.org/articulo.oa?id=309131077002, pp. 13–26 (2013)
2. Limón, M. R., Ortega, M. C.: Envejecimiento activo y mejora de la calidad de

vida en adultos mayores. Revista de Psicología y Educación, 1(6), pp. 225–238 (2011)
3. Trentin, G.: E-learning and the third age. Journal of Computer Assisted Learning, 20(1), 21–30, doi:10.1111/j.1365-2729.2004.00061.x (2004)
4. Moreno, M. D.: Alfabetización digital: el pleno dominio del lápiz y el ratón. Comunicar, 30, 137–146, doi:10.3916/c30-2008-02-007 (2008)
5. Naciones, U.: La sostenibilidad y la inclusión de las personas mayores en el entorno urbano. Disponible en: http://www.un.org/es/events/olderpersonsday (2015)
6. Alcantud, F.: Estudiantes con Discapacidades Integrados en los Estudios Universitarios: Notas para su Orientación. En: Rivas, F. (Ed.), Manual de Asesoramiento y Orientación Vocacional, Primera Edición, Síntesis, Madrid (1995)
7. Naciones, U.: Convención sobre los Derechos de las Personas con Discapacidad. Disponible en: http://www.un.org/esa/socdev/enable/documents/tccconvs.pdf (2015)
8. Johnson, L., Adams, B., Gago, D., Martín, S.: NMC Perspectivas Tecnológicas: Educación Superior en América Latina 2013-2018. Un Análisis Regional del Informe Horizon del NMC, Austin, Texas, The New Media Consortium (2013)
9. Gil, S.: Cómo hacer Apps Accesibles. Centro de Referencia Estatal de Autonomía Personal y Ayuda Técnica, Disponible en: http://www.ceapat.es/ ceapat_01/index.htm, pp. 5–85 (2013)
10. Cuello, J.: Diseñando apps para móviles. Disponible en: http://appdesignbook. com (2005)
11. Fuentes, D., Opitz, I., Oro, N.: Análisis de Aplicación Móvil basada en Geolocalización. Disponible en: http://profesores.elo.utfsm.cl/~agv/elo322/1s16/ projects/reports/Informe_G14_Elo322.pdf (2013)
12. Bernardos, A., Besada, J., Casar, C.: Tecnologías de localización, ETS Ingenieros de Telecomunicación. Universidad Politécnica de Madrid, Disponible en: http://www.upm.es/sfs/Rectorado/Organos%20de%20Gobierno/Consejo%20 Social/Actividades/tecnologias_servicios_para_sociedad_informacion.pdf (2005)
13. Nielsen, J., Norman, D.: The Definition of User Experience. Disponible en: http://www.nngroup.com/articles/definition-user-experience (2014)
14. Extreme Programming. Disponible en: http://www.extremeprogramming.org (2013)
15. Barry, D.: Service Architecture. Disponible en: http://www.service-architecture. com/articles/web-services/soap.html (2016)
16. Acuña, A., Romo, M.: Diseño Instruccional Multimedia. Pearson Education, México (2011)
17. Pino, M. R., Rodríguez, B., Soto, J. G.: Las Personas mayores y las TIC. Un compromiso para reducir la brecha digital. Pedagogía Social, Revista Interuniversitaria, doi:10.7179/PSRI_2015.26.13, 26, pp. 337–359 (2015)

ISSN 1870-4069

Ingeniería inversa en base a pruebas unitarias

L. Alberto Nicolás Ramírez[1], Carlos Perez-Corona[1], Yesenia N. González-Meneses[2]

Instituto Tecnológico de Apizaco, Departamento de sistemas y computación,
Tlaxcala, México

Facultad de Ciencias Básicas, Ingeniería y Tecnología, UAT,
Tlaxcala, México

{nic.lui.ram, cperezcorona}@gmail.com, yeseniaglez@hotmail.com

Resumen. Las metodologías proporcionan herramientas para llevar a cabo el correcto desarrollo de software buscando obtener software de calidad, en este sentido la base documental que describe a un sistema juega un papel importante al momento de dar mantenimiento, por ejemplo. Sin embargo, por malas prácticas es común encontrar sistemas sin esa base documental. Es allí donde surge la necesidad de crear una metodología que permita generar los productos base de la ingeniería de software. Este trabajo se enfoca a resolver el problema antes mencionado partiendo de pruebas unitarias utilizando software de testing y la interacción del usuario con el sistema o plataforma de caso, obteniendo un panorama general con los productos obtenidos, los diagramas UML obtenidos modelaran el sistema para ser comprendido e inferir los requerimientos iniciales, así como encontrar posibles errores, como pueden ser bugs, objetos o métodos no utilizados y detectar posibles riesgos generando problemas internos en la ejecución del sistema.

Palabras clave: Prueba del software, ingenieria de software, prueba de unidad, ingenieria inversa.

Reverse Inverse Engineering Based on Unit Tests

Abstract. The methodologies provide tools for the proper development of software with quality, in this sense the documental base that describes a system plays an important role at the moment of maintenance, for example. However, due to bad practices, it is common to find systems without this documental base. From here, arises the need to create a methodology that allows generating the base products of software engineering. This work focuses on solving the aforementioned problem starting from unit tests using testing software and the user's interaction with the system or platform of the case, the UML diagrams obtained will model the system to be understood and infer the requirements. As well as to find possible errors, such as bugs, unused objects or methods and to detect possible risks generating internal problems in the execution of the system.

Keywords: Software testing, software engineering, unit test, reverse software engineering.

1. Introducción

El desarrollo de software, se divide en fases que contemplan el análisis de requerimientos buscando conocer las necesidades del cliente, en base a ellas se modela una solución y posteriormente entrar a la fase de codificación, continuando así a la fase de pruebas; por cada fase debe generarse productos documentales. La fase de pruebas debe ser considerada por el equipo de desarrollo para asegurar la integridad del software cuando se encuentre en producción.

Este trabajo propone una metodología utilizando ingeniería inversa a partir de pruebas unitarias.

Para iniciar la metodología analiza las funciones del sistema (o plataforma) obteniendo los componentes claves que la integran y generando una base documental que ayudará a establecer parámetros de evaluación. Contando con los componentes de la plataforma se debe establecer las herramientas de evaluación idóneas para la ejecución de pruebas mediante frameworks de testing.

Habiendo realizado pruebas se obtendrán los objetos y actores que intervienen generando entonces los diagramas de casos de uso.

En base a los diagramas de caso de uso puede generarse los diagramas de clases y de secuencia.

Con la información obtenida se generará un documento mostrando los resultados obtenidos de las pruebas unitarias resaltando fallas, observaciones y métodos no utilizados; dicho documento será la referencia para resolver dichos problemas.

Por último, serán validados los requerimientos iniciales de la plataforma y se volverá a la parte inicial de la metodología o terminar el ciclo e validación.

2. Estado del arte

Software testing for object-oriented software.

La propuesta descrita en [1] busca mostrar los pasos para realizar pruebas de manera correcta, mostrando qué diagramas son necesarios para realizar dichas pruebas.

La creación de diagramas UML es el fuerte de esta propuesta, ya que enfatiza la necesidad de ellos para dar mayor precisión a los resultados.

Sin embargo no se busca generar documentación solo explica una sugerencia de pasos a seguir para hacer pruebas al software.

Attributes effecting software testing estimation; is organizational trust an issue?

La propuesta en [2] busca implementar correctas prácticas para el desarrollo de pruebas mediante reuniones para establecer dichos parámetros.

Si, bien las reuniones del equipo de software juegan un papel importante, esta propuesta hace notoria la necesidad de las mismas, esto con el objetivo de que el equipo

de desarrollo esté al tanto de los avances y estatus del proyecto y con esto disminuir la presencia de errores considerablemente.

La propuesta expresa como establecer métricas para evaluar software, sin embargo no busca generar diagramas de modelado del sistema.

A framework of the use of information in software testing. En [3] se busca entrar más en materia acerca de las pruebas de software, aquí se menciona la necesidad de realizar entrevistas al equipo de desarrollo para poder generar pruebas más exactas y comprobar la calidad del producto. Contar con información del software a desarrollar es importante para garantizar que el software se hará como fue solicitado y con eso evitar retrasos o trabajo de más en el desarrollo del mismo. Se menciona la generación de productos, pero no busca analizar el código fuente y solo se centra en pruebas unitarias.

3. Metodología

La propuesta de metodología que se muestra en la figura 1, se enfoca al análisis del algún software o sistema de información que no cuente con base documental, y que mediante una serie de pruebas unitarias pueda determinar los requerimientos iniciales y los productos de ingeniería de software del sistema, podrá verificarse si dicho sistema fue diseñado o codificado de acuerdo a los requerimientos. La metodología busca obtener los siguientes objetivos:

- Generar una metodología que en base a pruebas unitarias.
- Evaluar la plataforma utilizando pruebas unitarias.
- Identificar ventajas de frameworks de testing para evaluar la plataforma.
- Generar productos de las fases de análisis, diseño e implementación a partir de pruebas.
- Identificar a los actores que intervienen en la plataforma.
- Documentar de casos de uso.
- Documentar las pruebas expresándolas en diagramas de caso de uso.
- Representar la relación de los objetos entre si mediante diagramas de secuencia.
- Mostrar la estructura estática del sistema mediante diagramas de clases.
- Identificar posibles debilidades en la plataforma.
- Dar opciones de mejora en el desarrollo de software.

Análisis de la plataforma a evaluar para la comprensión de la misma. En esta fase se analizara el estado actual del sistema, mediante la observación y la interacción con la misma. Para ello es necesario entrevistar a quien interactúa con la plataforma a manera de conocer cómo funciona. Es necesario que el analista también interactúe con la plataforma para obtener una visión general del funcionamiento de la misma. Con una serie de preguntas y respuestas hechas por el analista al equipo de trabajo que utiliza la plataforma.

Análisis de frameworks adecuados para la evaluación. En esta fase es necesario determinar que Frameworks de testing que existen y en base al lenguaje de la

plataforma. Con ello se puede establecer el tipo de pruebas y obtener un mayor conocimiento del funcionamiento de la plataforma e identificar los actores y casos de uso de la que intervienen en la plataforma.

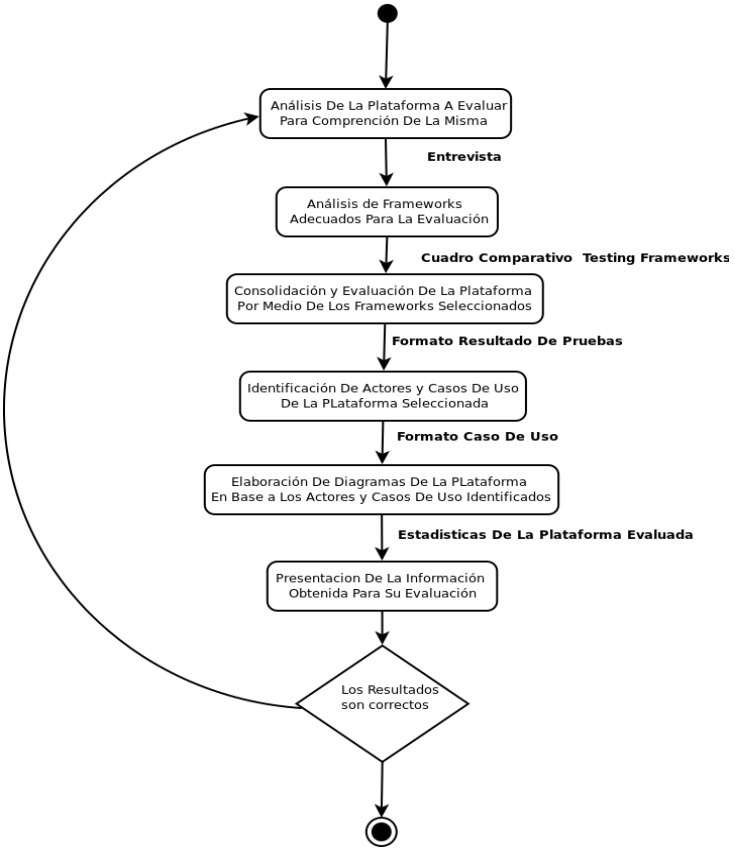


Fig. 1. Metodología de Ingeniería inversa en base a pruebas unitarias.

Consolidación y evaluación de la plataforma por medio de los frameworks seleccionados. En esta fase es deseable tener acceso al código fuente y la interfaz gráfica de la plataforma y analizando la función de cada uno de los componentes del sistema. Realizando pruebas unitarias para comprobar, si cada uno de los componentes del sistema realiza la función deseada. Por ejemplo el método A espera un valor entero resultado de una consulta, se debe poner a prueba dicho método ya sea enviando valores nulos o algún entero que sea igual a 0.

Identificación de actores y casos de uso de la plataforma seleccionada. En esta fase se ha adquirido un mayor conocimiento del funcionamiento y estructura de la plataforma seleccionada y es donde es necesario identificar a los actores que intervienen, estos pueden ser el usuario del sistema y pueden derivarse más actores, si

la plataforma seleccionada cuenta con roles de usuario, por ejemplo administrador e invitado.

Elaboración de diagramas de la plataforma en base a los actores y casos de uso identificados. En esta fase se tendrá una visión general del funcionamiento del sistema para generar diagramas de caso de uso e identificar como los usuarios (actores) y las acciones que realizan (casos de uso). Sin embargo es necesario demostrar la secuencia detallada de los diagramas de caso de uso, por lo que será necesario recurrir a los diagramas de secuencia por ultimo para mostrar la estructura estática del sistema se deben realizar diagramas de clases.

Presentación de la información obtenida. Última fase, donde en base a los diagramas y el resultado de las pruebas unitarias se puede inferir cuales son los requerimientos iniciales cotejando con la plataforma actual si estos son los correctos, obteniendo una visión general de la estabilidad y funcionalidad de la plataforma y poder dar propuestas de mejora si las hubiera.

4. Demostración de la metodología

Análisis De La Plataforma A Evaluar. Dentro de este apartado se mostrará los resultados del análisis de la plataforma utilizada como caso de estudio, que se encuentra en fase de producción, sin embargo, aun no se han realizado las pruebas pertinentes para evaluar la calidad del mismo. Considerando que no existe una base documental ni productos de ingeniería de software, no existe certeza que el resultado de las pruebas sea el esperado; por lo tanto fue necesario entrevistar al equipo de desarrollo y directivos de la empresa para que la información permita determinar que pruebas miden o evalúan la calidad de la plataforma.

Análisis de frameworks Adecuados para la evaluación. En esta fase se muestran los frameworks que serán utilizados para evaluar la plataforma. A continuación serán explicadas las ventajas y desventajas de los frameworks seleccionados:

Arquillian. Mediante el objeto **ShrinkWrap** permite construir el tipo de empaquetado del proyecto que se esté utilizando (.jar, .war etc.) y ejecutar el test, está característica permite trabajar en conjunto con servidores de aplicaciones como GlassFish o JBoss [5] y utilizar el criterio de persistencia para realizar conexión a un servidor de base de datos como mysql [6] o SQL Server [7] *Arquillian* utiliza librerías de JUnit y tiene soporte con JPA y se puede contemplar como una opción viable para implementar el testing.

Selenium. Este framework guarda los datos que fueron digitados al realizar dichas pruebas. *Selenium* registra el evento y datos digitados por el usuario y los valores que fueron digitados o generados por el evento se registran en la interfaz del programa.

Consolidación y evaluación de la plataforma por medio de los frameworks seleccionados. Los criterios de las pruebas efectuadas a la plataforma serán mostradas en esta sección.

Test Unitario: Arquillian permite crear instancias objetos de clase y por ende invocar sus métodos, de tal modo que sea posible realizar un "deploy" de la clase en forma aislada resultando en una validación de resultados y comportamientos.

Niveles de Pruebas. Las pruebas realizadas en la plataforma se dividen en dos fases, unitarias, funcional e integración, esto con la finalidad (o propósito) de realizar un análisis adecuado. [4].

Test Funcional e Integración: Se reproducirán las pruebas grabadas previamente con una serie de parámetros diferentes con la finalidad de evaluar la comunicación de la interfaz gráfica (front end) con los componentes de más bajo nivel (back end).

Parámetros de Evaluación. Para determinar sí las pruebas realizadas a la plataforma cumplen con los criterios esperados, garantizando la calidad de los componentes, se realizan evaluaciones en base a la salida obtenida y avalados por los líderes de proyecto.

Dichos parámetros son formados por los siguientes elementos:

- Reunir los datos básicos de un caso de uso.
  - Nombre Del Requisito Funcional
  - Versión
  - Objetos Asociados
- Descripción
  - Pre-Condición: mostrará qué acción debe ser invocada o ejecutada para utilizar el caso o casos de uso.
  - La secuencia de los eventos y qué casos de uso intervienen.
  - Post-Condición: mostrará el resultado esperado cuando se termine la ejecución del caso de uso.
  - Excepciones
  - Rendimiento
  - Importancia
  - Urgencia
  - Comentarios

Identificación de actores y casos de uso de la plataforma seleccionada. En esta fase se obtienen los actores que intervienen en la plataforma. En nuestro ejemplo, los actores identificados son los de la tabla 2.

Casos de uso. De la entrevista realizada al equipo de desarrollo se obtienen las funciones de la plataforma. Siendo identificados casos de uso y actores se procede a crear los diagramas UML para modelar la plataforma, en este caso sólo se explicará con el resultado de un diagrama de casos de uso: Administración de Usuarios.

Tabla 1. Identificación de actores.

Nombre Actor	Descripción	Tipo
Usuario Administrador	Usuario que cuenta con el acceso a todas las funciones de la plataforma	Persona
Usuario General	Representación de las funciones que, los usuarios comparten en común	Persona
Sistema	plataforma	Sistema

Elaboración de diagramas en base a los actores y casos de uso identificados. En esta fase se mostrará los diagramas de casos de uso, de clases y secuencia generados. En base a los diagramas de caso de uso generados que fueron analizados se tiene como resultado obtener diagramas de secuencia del caso diagrama anterior generando diagramas de secuencia y diagramas de clase, dichos diagramas modelaran el módulo o función analizada.

Para un mejor entendimiento se explicara el diagrama utilizando el patrón de diseño MVC (Modelo-Vista-Controlador). A continuación se explica un módulo del sistema analizado.

Administrador de Usuarios. Una de las funciones principales del sistema es el módulo de usuarios, donde se muestra quien puede acceder al sistema y los permisos con los que cuenta. Este módulo permite crear, editar, eliminar y consultar los usuarios registrados, para acceder a este módulo es necesario contar con permisos de administrados o supervisor. De donde se obtuvieron los diagramas que se ilustran en las figuras 2, 3 y 4. La figura 2 muestra cómo se obtuvieron los casos de uso utilizando la información contenida en el formato que se describe en la figura 3.

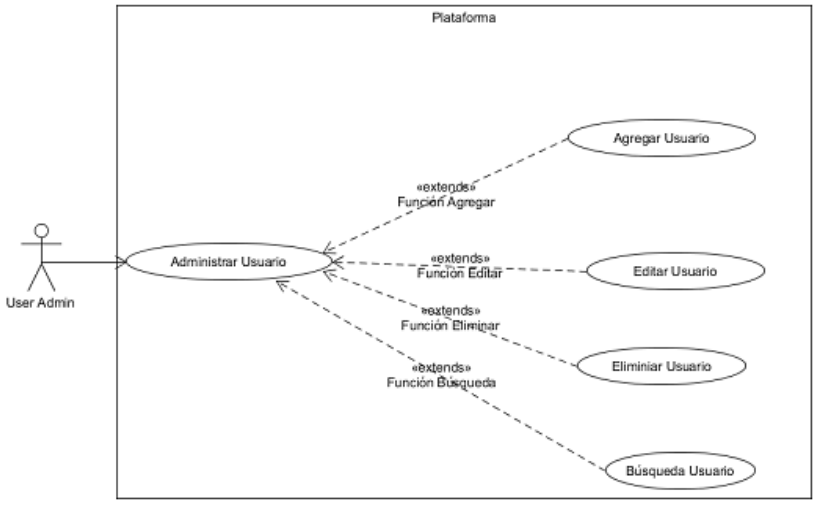


Fig. 2. Diagrama caso de uso administrador de usuarios.

RF- <id del requisito>	User-Admin.xhtml	001
Versión	0.3	
Autores		
Fuentes		
Objetivos asociados	Index.xhtml	
Descripción	Administración de usuarios, muestra los datos básicos de los usuarios y permite editar, agregar y asignar usuarios.	
Precondición	Haber iniciado cuenta de administrador.	
Secuencia Normal	**Paso**	**Acción**
	1	El *usuario* **selecciona** *(load)*el módulo de usuarios.
	2	El *sistema* **muestra** (*readAll)* los usuarios activos registrados.
	3	El *usuario* **selecciona** <<función>>
	4	El *sistema* **ejecuta** <<función>>
Pos condición	Guardar cambios realizados	
Excepciones	**Paso**	**Acción**
	2.1	El sistema no muestra resultados por error de conexión
	4.1	El sistema no ejecuta la función seleccionada.
Importancia	Vital	
Urgencia	Urgente	
Casos de uso extensión	Add-User Edit-User Delete-User Busqueda-User	

Fig. 3. Formato de caso de uso administrador de usuarios.

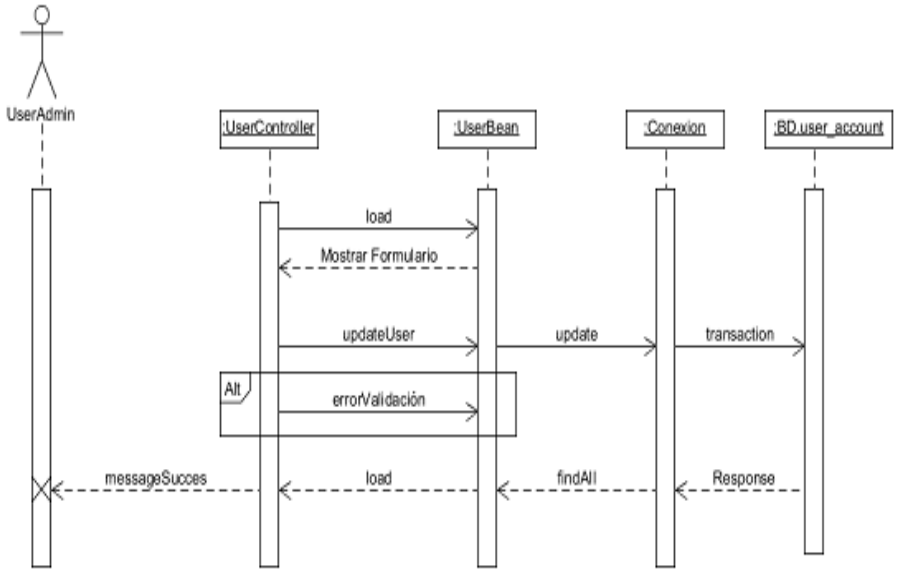


Fig. 4. Diagrama de secuencia administrador de usuarios.

Presentación de la información obtenida. Esta fase muestra al equipo de desarrollo la información obtenida del caso de uso (User-Admin) es la que se presenta en la tabla 2, esta información debe ser utilizada para mejorar la plataforma o sistema, la inferencia de los requerimientos se dan al analizar los diagramas obtenidos.

Productos obtenidos. Al realizar la metodología se obtienen los productos mostrados en la tabla 3, los cuales son necesarios para inferir los requerimientos iniciales o buscar errores y opciones de mejora.

Tabla 2. Información obtenida.

Observaciones	Errores	Total
Métodos que no manejan excepciones	161	346
Métodos que retornan null alguna query	221	346
Errores y observaciones encontradas	53	346
Métodos no utilizados	100	346

Tabla 3. Productos obtenidos.

Fase	Producto
Análisis de la plataforma a evaluar para la comprensión de la misma	Entrevista equipo desarrollo
Análisis de frameworks,adecuados para la evaluación	Cuadro comparativo de frameworks testing
Consolidación y evaluación de la plataforma por medio de los frameworks seleccionados	Formato resultado de pruebas
Identificación de actores y casos de uso de la plataforma seleccionada	Formatos de caso de uso
Elaboración de diagramas de la plataforma en base a los actores y casos de uso identificados	Estadísticas de la plataforma evaluada
Presentación de la información obtenida	

5. Conclusion y trabajos futuros

La ejecución de pruebas para el desarrollo de software es imprescindible, ya que la parecencia de errores en la fase de producción puede presentar un retraso significativo en la entrega del producto. Inicialmente se requería tener interacción con la plataforma y el equipo de desarrollo, debido a que era necesario conocer cómo funcionaría la plataforma, posteriormente se diseñaron pruebas con un framework de testing previamente analizadas donde se comprobó la funcionalidad de dicha plataforma. Como resultado de las pruebas unitarias se generaron productos de ingeniería de software como: actores, diagramas de caso de uso, diagramas de clase, diagrama entidad-relación y diagramas de secuencia. Terminada la metodología se mostraron los resultados al equipo de desarrollo haciendo propuestas de mejora si las hubiese.

Trabajos futuros. Para la mejora del trabajo antes expuesto es necesario implementar nuevas características a la metodología, las cuales serán mencionadas:

- Crear más diagramas UML para dar mayor exactitud a la metodología y que estos cambien en función a la arquitectura de la plataforma a evaluar.
- Adaptar la metodología para que sea compatible con la reingeniería de software, que en base al diseño de pruebas pueda ser interpretada como ingeniería inversa.
- En base al acceso que se requiere al código fuente se necesita hacer propuestas de optimización de código, por ejemplo, hacer reducción e identificación de variables públicas o privadas que no sean utilizadas y que puedan generar fugas de memoria.
- De acuerdo al patrón de diseño con el que fue diseñada la plataforma permitir la adaptación de la metodología.
- Implementar pruebas de sistema e integración o diseñar pruebas propias para dar mayor exactitud al resultado final.
- Permitir que la metodología sea compatible con mas arquitecturas de software.
- Suite dinámica que en base a la interacción del equipo de desarrollo genere los diagramas de la metodología.

Referencias

1. Biju, S. M.: Model-based software testing for object-oriented software. E-Learning, no. 5, pp. 885–891 (2008)
2. DuBois, P.: Mysql 5.0 reference manual. Disponible en: http://dev.mysql.com/ doc/refman/ 5.0/en/index.html (2015)
3. Hammoud, W.: Attributes effecting software testing estimation; is organizational trust an issue? Phoenix, ProQuest LLC, D.M.IST, Dissertation, University of Phoenix (2014)
4. it Mentor: Capacitación y guía para el desarrollo de software. Disponible en: http://materias.fi.uba.ar/7548/PruebasSoftware.pdf (2015)
5. JBoss: About jboss. Disponible en: http://www.jboss.org (2015)
6. Microsft: Sql server. Disponible en: http://www.microsoft.com/es-es/server-cloud/products/sql-server (2015)
7. Payman, K.: A framework of the use of information in software testing. Proquest LLC, D.Sc., Dissertation, The George Washington University (2010)

ISSN 1870-4069

Implementación de una metodología de ingeniería de requerimientos en grandes proyectos de desarrollo de software

Diego Armando Valles Montalvo, Perfecto Malaquías Quintero Flores, Higinio Nava Bautista

Tecnológico Nacional de México, Instituto Tecnológico de Apizaco, Tlaxcala, México

armen0621@gmail.com, kmalakof@yahoo.fr, higinionava@hotmail.com

Resumen. Actualmente no se puede negar el gran crecimiento del mercado sobre los productos de software, es ahí donde surge el interés por asegurar el éxito de dichos productos y la calidad en su desempeño y algo muy importante que cumpla con las necesidades del cliente como usuario final. En este artículo se analizará el proceso de Ingeniería de requerimientos que muchas veces se da por alto en el proceso de desarrollo de software y su gran influencia en el éxito del software final. Es por ello que tras describir la metodología y siendo aplicada a un proyecto de desarrollo de software para la administración de patios de la industria automotriz, en el presente documento se discuten los resultados obtenidos en la implementación de una metodología de ingeniería de requerimientos en el inicio del proceso de desarrollo de un proyecto de software.

Palabras clave: Requerimientos, ingeniería de requerimientos (IR), ingeniería del software, industria automotriz.

Implementation of a Methodology of Requirements Engineering in Big Software Development Projects

Abstract. At present, it is not possible to deny the great growth of the market on software products, it is there where the interest arises to ensure the success of these products and the quality of their performance and something very important that meets the needs of the customer as an end user. This paper will analyze the requirements engineering process that is often overlooked in the software development process and its great influence on the success of the final software. It is for this reason that after describing the methodology and being applied to a software development project for the administration of patios of the automotive industry, the present document discusses the results obtained in the implementation of a methodology of requirements engineering in the beginning of the development process of a software project.

Keywords: Requirements, requirements engineering, automotive industry.

1. Introducción

Las actividades de diseño y construcción de software de computadoras son desafiantes, creativas y hasta divertidas. La construcción es tan irresistible que muchos desarrolladores de software quieren entrar en ella antes de comprender con claridad de qué es lo que se necesita.

La comprensión de los requerimientos de un problema está entre las tareas más difíciles que enfrenta un ingeniero de software. Cuando se piensa por primera vez acerca de ello, la Ingeniería de requerimientos no parece tan difícil.

Los requerimientos para un sistema son descripciones de lo que el sistema debe hacer: el servicio que ofrece y las restricciones en su operación, tales requerimientos reflejan las necesidades de los clientes por un sistema que atienda cierto propósito, como sería controlar un dispositivo, colocar un pedido o buscar información, por lo tanto al proceso de descubrir, analizar, documentar y verificar estos servicios y restricciones se le llama ingeniería de requerimiento (IR).

1.1. Ingeniería del software

La Computer Society de la IEEE define la ingeniería del software como: "Aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software, es decir, la aplicación de la ingeniería al software".

La ingeniería del software es importante por dos razones para Sommerville [1], siendo estás las siguientes:

1. Cada vez con mayor frecuencia, los individuos y la sociedad se apoyan en los avanzados sistemas de software. Por ende, se requiere producir económica y rápidamente sistemas confiables.
2. A menudo resulta más barato a largo plazo usar métodos y técnicas de ingeniería del software para los sistemas de software, que sólo diseñar los programas como si fuera un proyecto de programación personal. Para muchos tipos de sistemas, la mayoría de los costos consisten en cambiar el software después de ponerlo en operación.

Según Sommerville [1] explica que existen cuatro actividades fundamentales que son comunes a todos los procesos de software, y éstas son:

1. Especificación del software, donde clientes e ingenieros definen el software que se producirá y las restricciones en su operación.
2. Desarrollo del software, donde se diseña y programa el software.
3. Validación del software, donde se verifica el software para asegurar que sea lo que el cliente requiere.
4. Evolución del software, donde se modifica el software para reflejar los requerimientos cambiantes del cliente y del mercado.

Para Pressman [2] la Ingeniería del software es una tecnología estratificada, como se muestra en la Fig. 1, cualquier enfoque de la ingeniería debe ser sustentado en el compromiso con la calidad. La gestión de la calidad total, sigma seis y enfoques similares fomentan una cultura de mejora continua del proceso, y es esta cultura la que al final conduce al desarrollo de enfoques muy efectivos para la Ingeniería del software, la base que soporta la Ingeniería del software es un enfoque en la calidad.

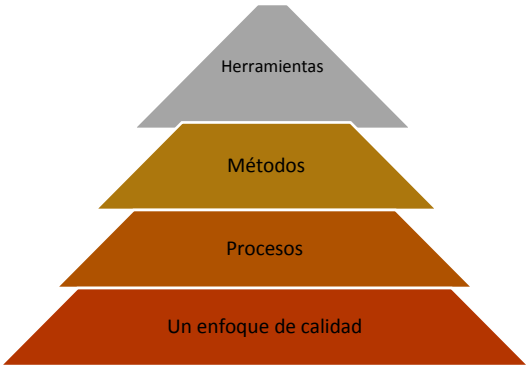


Fig. 1. Estrados de la Ingeniería del software.

La base de la ingeniería del software es el estrado del proceso, el proceso define un marco de trabajo que debe establecerse para la entrega efectiva de la tecnología de la ingeniería del software. El proceso de software forma la base para el control de la gestión de los proyectos de software y establece el contexto en el cual se aplican los métodos técnicos, se generan los productos del trabajo, se establecen los fundamentos, se asegura la calidad.

Los métodos de la ingeniería del software proporcionan los "cómo" técnicos para construir software, los métodos abarcan un amplio espectro de tareas que incluyen la comunicación el análisis de requerimientos, el modelo del diseño, la construcción del programa, la realización de pruebas y soporte.

Las herramientas de la ingeniería del software proporcionan el soporte automatizado o semiautomatizado para el proceso y los métodos, cuando las herramientas se integran de forma que la información que cree una de ellas pueda usarla otra, se dice que se ha establecido un sistema para el soporte del desarrollo del software.

1.2. Requerimientos

El INTECO (Instituto Nacional de Tecnologías de la Comunicación) explica que los mejores productos, desde el punto de vista del usuario, son aquellos creados por desarrolladores que tienen muy claro lo que se pretende conseguir con el producto y cómo obtenerlo. Para llegar a este punto, se debe entender el trabajo del usuario, cómo afectará el producto a su trabajo y cómo se adecuará a los objetivos de la organización.

Un requerimiento es una característica del sistema o una descripción de algo que el sistema es capaz de hacer con el objeto de satisfacer el propósito del sistema para

Pfleeger [3], lo que ha sido apropiadamente documentado y validado por el solicitante Berenbach [4]. Los requerimientos tratan exclusivamente sobre los fenómenos del dominio de aplicación y no sobre la máquina que los implementa Jackson [5].

Actualmente el mercado de productos de software ha incrementado considerablemente, es por ello que se debe de tener mayor atención análisis y su desarrollo para que así puedan cumplir con su propósito el cual es satisfacer las necesidades de un cliente. Para incrementar el grado de éxito se debe empezar con un buen levantamiento, análisis y diseño de requerimientos para que sirvan de base para los productos de software finales.

Hay dos formas de definir la Ingeniería de requerimientos según menciona Sommerville [1] y son:

1. Al proceso de descubrir, analizar, documentar y verificar estos servicios y restricciones.
2. El proceso para establecer los servicios que el sistema debería proveer y las restricciones bajo las cuales debería operar y ser desarrollado.

1.3. Procesos de ingeniería de requerimientos

Los procesos de la Ingeniería de requerimientos incluyen cuatro actividades de alto nivel. Éstas se enfocan en valorar si el sistema es útil para la empresa (estudio de factibilidad), descubrir requerimientos (adquisición y análisis), convertir dichos requerimientos en alguna forma estándar (especificación) y comprobar que los requerimientos definan realmente el sistema que quiere el cliente (validación).

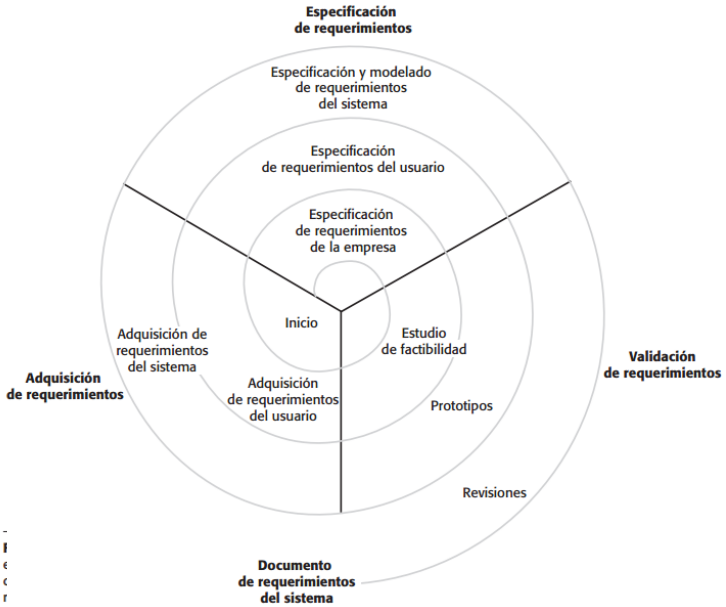


Fig. 2. Vista en espiral del proceso de Ingeniería de requerimientos.

La Fig. 2, presenta el entrelazamiento entre las diferentes fases del proceso de Ingeniería de requerimientos. Las actividades están organizadas como un proceso iterativo alrededor de una espiral, y la salida es un documento de requerimientos del sistema, la cantidad de tiempo y esfuerzo dedicados a cada actividad en cada iteración depende de la etapa del proceso global y el tipo de sistema que está siendo desarrollado. En el inicio del proceso, se empleará más esfuerzo para comprender los requerimientos empresariales de alto nivel y los no funcionales, así como los requerimientos del usuario para el sistema. Más adelante en el proceso, en los anillos exteriores de la espiral, se dedicará más esfuerzo a la adquisición y comprensión de los requerimientos detallados del sistema.

1.4. Adquisición y análisis de requerimientos

Sommerville [1] dice que después de un estudio de factibilidad inicial, la siguiente etapa del proceso de Ingeniería de requerimientos es la adquisición y el análisis de requerimientos. En esta actividad, los ingenieros de software trabajan con clientes y usuarios finales del sistema para descubrir el dominio de aplicación, qué servicios debe proporcionar el sistema, el desempeño requerido de éste, las restricciones de hardware, etcétera.

Fig. 3. El proceso de adquisición y análisis de requerimientos.

En la Fig. 3, se muestra un modelo del proceso de adquisición y análisis. Cada organización tendrá su versión o ejemplificación de este modelo general, dependiendo de factores locales, tales como experiencia del personal, tipo de sistema a desarrollar, estándares usados, etcétera.

Las actividades del proceso son:

– Descubrimiento de requerimientos. Éste es el proceso de interactuar con los participantes del sistema para descubrir sus requerimientos. También los

requerimientos de dominio de los participantes y la documentación se descubren durante esta actividad.

- Clasificación y organización de requerimientos. Esta actividad toma la compilación no estructurada de requerimientos, agrupa requerimientos relacionados y los organiza en grupos coherentes. La forma más común de agrupar requerimientos es usar un modelo de la arquitectura del sistema, para identificar subsistemas y asociar los requerimientos con cada subsistema. En la práctica, la Ingeniería de requerimientos y el diseño arquitectónico no son actividades separadas completamente.
- Priorización y negociación de requerimientos. Inevitablemente, cuando intervienen diversos participantes, los requerimientos entrarán en conflicto. Esta actividad se preocupa por priorizar los requerimientos, así como por encontrar y resolver conflictos de requerimientos mediante la negociación. Por lo general, los participantes tienen que reunirse para resolver las diferencias y estar de acuerdo con el compromiso de los requerimientos.
- Especificación de requerimientos. Los requerimientos se documentan e ingresan en la siguiente ronda de la espiral. Pueden producirse documentos de requerimientos formales o informales.

1.5. Validación de los requerimientos

La validación de requerimientos es el proceso de verificar que los requerimientos definan realmente el sistema que en verdad quiere el cliente. Se traslapa con el análisis, ya que se interesa por encontrar problemas con los requerimientos. La validación de requerimientos es importante porque los errores en un documento de requerimientos pueden conducir a grandes costos por tener que rehacer, cuando dichos problemas se descubren durante el desarrollo del sistema o después de que éste se halla en servicio Sommerville [1].

Hay algunas técnicas de validación de requerimientos que se usan individualmente o en conjunto con otras:

- Revisiones de requerimientos. Los requerimientos se analizan sistemáticamente usando un equipo de revisores que verifican errores e inconsistencias.
- Creación de prototipos. En esta aproximación a la validación, se muestra un modelo ejecutable del sistema en cuestión a los usuarios finales y clientes. Así, ellos podrán experimentar con este modelo para constatar si cubre sus necesidades reales.
- Generación de casos de prueba. Los requerimientos deben ser comprobables. Si las pruebas para los requerimientos se diseñan como parte del proceso de validación, esto revela con frecuencia problemas en los requerimientos. Si una prueba es difícil o imposible de diseñar, esto generalmente significa que los requerimientos serán difíciles de implementar, por lo que deberían reconsiderarse.

1.6. Administración de los requerimientos

Los requerimientos para los grandes sistemas de software siempre cambian. Una razón es que dichos sistemas se desarrollaron por lo general para resolver problemas "complejos": aquellos problemas que no se pueden definir por completo. Como el problema no se logra definir por completo, los requerimientos del software están condenados también a estar incompletos. Durante el proceso de software, la comprensión que los participantes tienen de los problemas cambia constantemente ver Fig. 4. Entonces, los requerimientos del sistema también deben evolucionar para reflejar esa visión cambiante del problema.

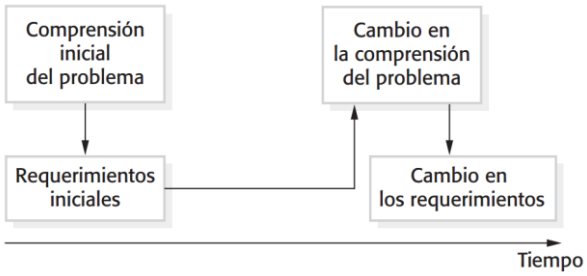


Fig. 4. El proceso de adquisición y análisis de requerimientos.

2. Estado del arte

La ingeniería de requerimientos es la fase más eficaz del proceso de desarrollo de software. Su objetivo es recoger buenos requerimientos de los interesados en la forma correcta. Es importante por cada organización para desarrollar productos de software de calidad que puede satisfacer las necesidades del usuario.

La ingeniería de requerimientos para el proceso de desarrollo de software es un ejercicio complejo que considera producto demandas de un gran número de puntos de vista, papeles, responsabilidades y objetivos. Por lo tanto, se hace necesario aplicar prácticas de ingeniería requerimiento en todas las fases de proceso de desarrollo de software Dhirendra [6].

A. Charan Kumari [7] hace mención que en la fase de ingeniería requerimientos del ciclo de vida de desarrollo de software, uno de las principales preocupaciones de los ingenieros de software es seleccionar un conjunto de requerimientos de software para su implementación en la próxima versión del software de muchos requerimientos propuestos por los clientes, mientras que el equilibrio del presupuesto y la satisfacción del cliente.

El autor Philipp Holtkamp [8] presenta que el enfoque más común es escribir los requerimientos de usuario mediante el lenguaje natural. La ventaja es que el lenguaje natural es el principal medio de comunicación entre las partes interesadas. Sin embargo, problemas como la imprecisión, las incomprensiones, la ambigüedad y la incoherencia son comunes cuando se utiliza el lenguaje natural.

Sin embargo, el lenguaje natural no es estructurado ni formal ni gráfico, y puede ser demasiado orientado a los algoritmos y lenguajes de programación concretos. Un conocido diagrama utilizado para el modelado de requerimientos son los casos de uso. Incluso antes de UML surgió como el principal lenguaje de modelado de la ingeniería del software.

El autor Michel dos Santos Soares [9] discute que el modelado debe proporcionar los medios para expresar las necesidades gráficas. Los modelos gráficos comunes pueden facilitar la comunicación de los modelos para los interesados. Los modelos deben ser legible para el ser humano, como los múltiples actores involucrados tienen que comprender los modelos. En este caso, el equilibrio es necesario, como por ejemplo los requerimientos más legibles por la máquina, menos legible se vuelven para el por el ojo humano.

3. Implementación de la metodología de ingeniería de requerimientos

Para la adquisición de requerimientos se implementaron cuestionarios y entrevistas como herramientas de recolección de datos de información como lo menciona la metodología, tras la obtención de la información se hizo el análisis de la misma para la generación de requerimientos de software, por medio de la clasificación de requerimientos funcionales y no funcionales.

Se definió una nomenclatura para la clasificación de los requerimientos una vez definidos y priorizados por el nivel de importancia e impacto en el desarrollo del sistema, dejando así definidas las relaciones entre requerimientos y separando los requerimientos que son determinados como de menor impacto. Al realizar la clasificación definimos en una escala del 1 al 10 la importancia y su grado de dificultad al ser desarrollado, y así determinar el tiempo que tomara para ser concluido.

Logrando así la obtención de los primeros requerimientos informales para su negociación con el cliente disipando diferencias negativas para futuras revisiones.

Mediante una reunión de revisión de avance se validaron los requerimientos definidos por todo el equipo de trabajo y el cliente, por medio de esto se puede verificar si lo propuesto satisface con las necesidades del cliente recolectadas inicialmente en el proyecto. Se logró encontrar errores de análisis y priorización evitando así costos innecesarios en el futuro de proyecto.

Para una mejor representación de los requerimientos establecidos se elaboraron los prototipos de bajo, medio y alto nivel para la representación gráfica de los requerimientos y llevar al cliente a una conceptualización más sólida sobre el análisis que se realizó al problema propuesto.

A todo este proceso se elaboraron los documentos que sustenten los cambios y las entregas que se realizaron, que sirven de apoyo a futuras explicaciones y para dejar como acuerdos de negociación evitando así futuros mal entendidos y teniendo documentación base para el desarrollo del sistema.

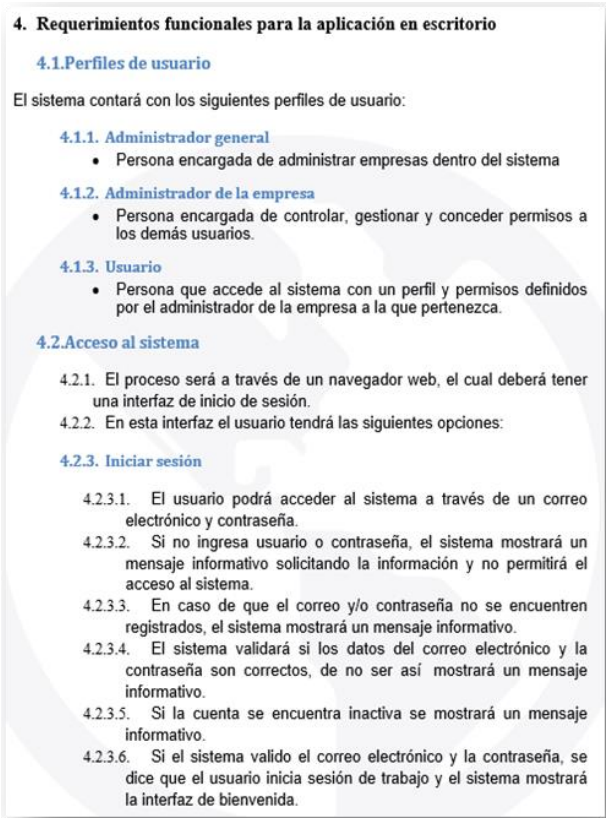
4. Resultados experimentales

Tomando como caso experimental el desarrollo de un proyecto de software para la "Administración de patios de la industria automotriz", se implementó el proceso que propone la Ingeniería de requerimientos, y se obtuvieron los siguientes resultados:

4.1. Especificación de requerimiento de software (ERS)

En la Fig. 5, se muestra un extracto del documento ERS donde se describen los requerimientos ordenándolos de manera secuencial y por jerarquías de dependencias, logrando así su interpretación a la hora de realizar el diseño y codificación del software.

Una de las tareas más importante de la ingeniería de requerimientos es la recolección y análisis de los requerimientos, es una tarea ardua entre una serie de métodos para la recolección es por ello que nos dimos a la tarea de aplicar entrevistas y sesiones de pláticas con el cliente para así juntar la información necesaria y tras un análisis elaborar un documento formal como lo es el ERS que sirve como evidencia de conformidad por parte del cliente.



4. Requerimientos funcionales para la aplicación en escritorio

4.1.Perfiles de usuario

El sistema contará con los siguientes perfiles de usuario:

4.1.1. Administrador general

- Persona encargada de administrar empresas dentro del sistema

4.1.2. Administrador de la empresa

- Persona encargada de controlar, gestionar y conceder permisos a los demás usuarios.

4.1.3. Usuario

- Persona que accede al sistema con un perfil y permisos definidos por el administrador de la empresa a la que pertenezca.

4.2.Acceso al sistema

4.2.1. El proceso será a través de un navegador web, el cual deberá tener una interfaz de inicio de sesión.

4.2.2. En esta interfaz el usuario tendrá las siguientes opciones:

4.2.3. Iniciar sesión

4.2.3.1. El usuario podrá acceder al sistema a través de un correo electrónico y contraseña.

4.2.3.2. Si no ingresa usuario o contraseña, el sistema mostrará un mensaje informativo solicitando la información y no permitirá el acceso al sistema.

4.2.3.3. En caso de que el correo y/o contraseña no se encuentren registrados, el sistema mostrará un mensaje informativo.

4.2.3.4. El sistema validará si los datos del correo electrónico y la contraseña son correctos, de no ser así mostrará un mensaje informativo.

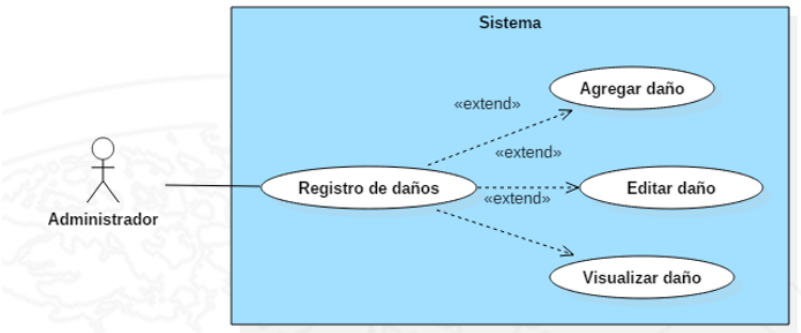
4.2.3.5. Si la cuenta se encuentra inactiva se mostrará un mensaje informativo.

4.2.3.6. Si el sistema valido el correo electrónico y la contraseña, se dice que el usuario inicia sesión de trabajo y el sistema mostrará la interfaz de bienvenida.

Fig. 5. Especificación de Requerimientos de Software.

4.2. Diagrama de casos de uso

Un análisis de funcionalidad como primera instancia del análisis de los requerimientos da como resultado la elaboración de los casos de uso, para conocer así las principales interacciones que tendrá el usuario con el software, ver Fig. 7.



ID - UC:	Caso de uso 05
Nombre de UC:	Registro de daños

Actores:	Administrador de empresa y/o usuario del sistema
Descripción:	En éste módulo se permite administrar los daños que se

Fig. 7. Diagrama de casos de uso.

Los casos de uso siento uno de los diagramas típicos que se elaboran tras iniciar el análisis de los requerimientos, quedo reflejado en este proyecto ya que podemos discernir las interacciones del usuario así como los posibles escenarios en los cuales se puede encontrar el usuario dentro del software.

4.3. Diagrama de actividades

Los diagramas de actividades nos permitieron ver reflejada de manera el flujo de las actividades del usuario dentro del producto de software, ver Fig. 8. A grandes rasgos el usuario realiza una serie de actividades dentro de software las cuales se reflejaron en la elaboración de diagramas de secuencia, para realizar de manera visual las posibles actividades que el usuario realizara y mostrar así un flujo de acciones.

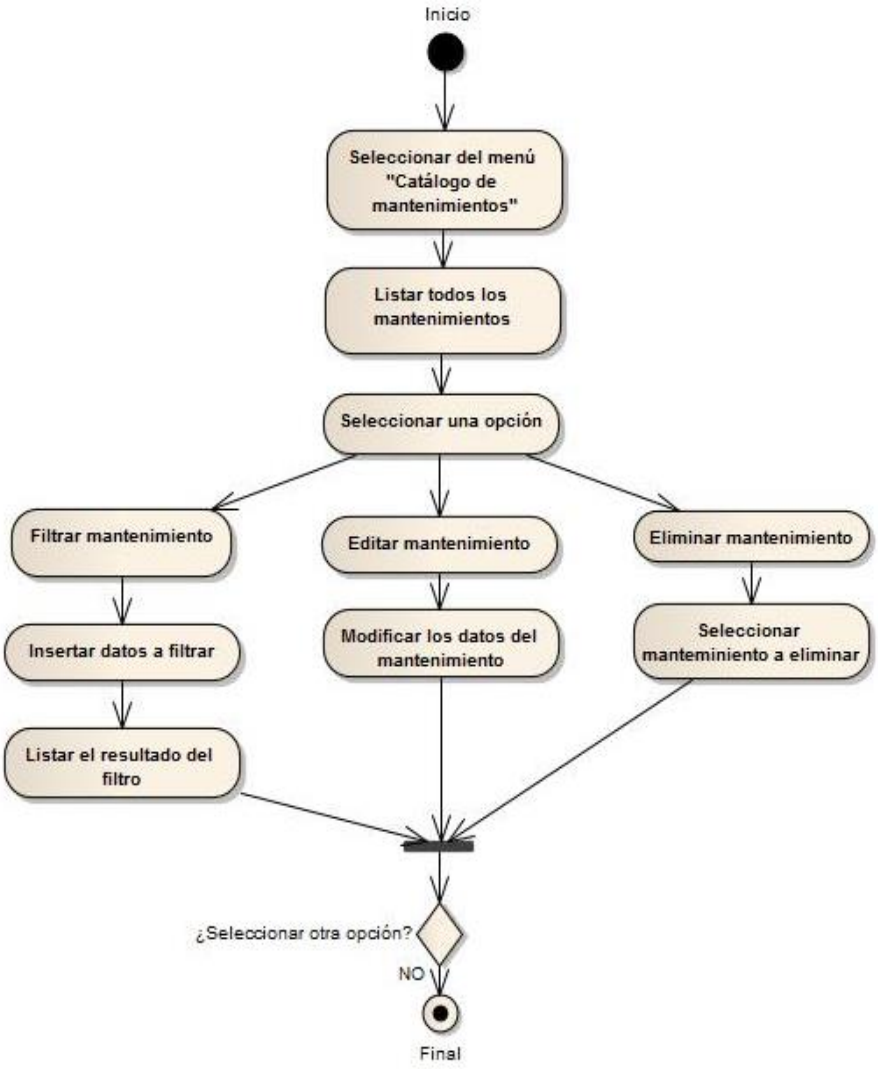


Fig. 8. Diagrama de actividades.

4.4. Diagrama de procesos BPM

En la Fig. 9, vemos el resultado de un diagrama BPM que sirve para ejemplificar el proceso de interacción de un usuario dentro del producto de software, permitiendo así la retroalimentación entre usuario y analista.

El flujo de procesos que refleja el diagrama de procesos BPM, nos da un marco referencial más exacto del funcionamiento general del software, en este caso se dividió por módulos de acciones y a su vez en submódulos, y por como resultado se obtuvo un análisis más a fondo asegurando la viabilidad del proyecto.

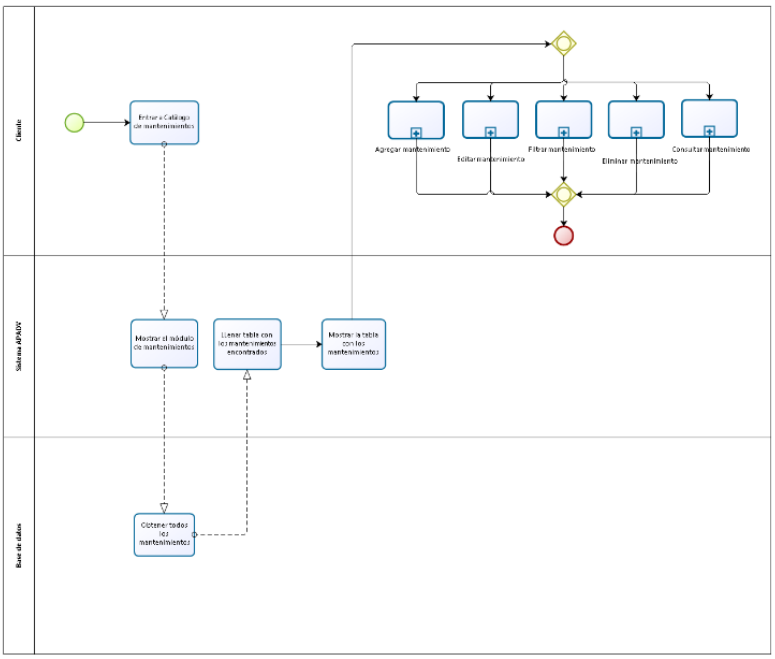


Fig. 9. Diagrama de procesos BPM.

4.5. Prototipos

Actualmente la creación de prototipos hacen más fácil la tarea de recolección de requerimientos asegurando que lo entendido hasta el momento por el analista es lo que realmente el usuario necesita, ver Fig. 10.

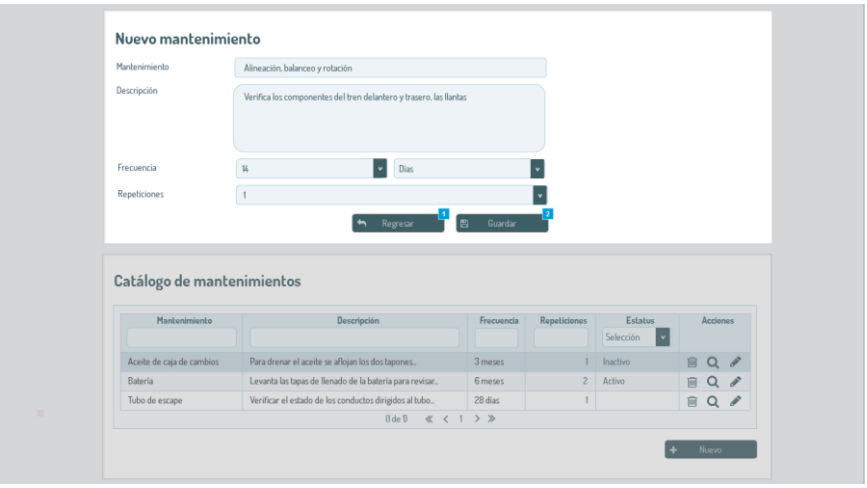


Fig. 10. Prototipo de alto nivel.

Los prototipos elaborados fueron un plus total al proyecto generando mayor aceptación por parte del cliente el cual de manera inmediata pudo ver reflejada las respuestas a su lista de necesidades.

5. Conclusiones y trabajo futuro

Se concluye que la implementación adecuada de la ingeniería de requerimientos al inicio de proyectos para la elaboración de software nos permite identificar y entender de manera correcta las necesidades de los clientes, dando como resultado la elaboración del software a la medida de sus requerimientos, y con esto aseguramos el éxito de la aceptación del producto de software por parte de usuario final.

Para las investigaciones futuras, se propone el diseño de un nuevo proceso de Ingeniería de requerimientos rescatando las actividades de mayor impacto y proponiendo nuevas actividades dentro del proceso, aportando así un proceso funcional para proyectos de software generales.

Agradecimientos. Queremos agradecer a todos los impulsores que trabajan a favor de mejorar el desarrollo de software.

Referencias

1. Sommerville, I.: Ingeniería del software. Estado de México, Pearson (2011)
2. Pressman, R. S.: Ingeniería del software. Un enfoque práctico. México, MacGraw Hill (2005)
3. Pfleeger, S. L.: Software Engineering: Theory and Practice. NJ, USA, Upper Saddler River (2001)
4. Brian, D. P.: Software & Systems Requirements Engineering: In Practice. Inc., New York, NY, USA, McGraw-Hill (2009)
5. Jackson, M.: Software requirements & specifications: a lexicon of practice, principles and prejudices. New York, NY, USA, ACM Press/Addison-Wesley Publishing Co. (1995)
6. Dhirendra, U. S.: An Effective Requirement Engineering Process Model for Software Development and Requirements Management. In: International Conference on Advances in Recent Technologies in Communication and Computing (2010)
7. Charan, K. S.: Comparing the performance of quantum-inspired evolutionary algorithms for the solution of software requirements selection problem. Elsevier (2016)
8. Holtkamp, P.: Soft competency requirements in requirements engineering, software design, implementation and testing. Elsevier (2014)
9. dos Santos, M, Vrancken, J.: User Requirements modeling and analysis of software-intensive systems. Elsevier (2010)

ISSN 1870-4069

Futbol: lenguaje formal y simulación computable

Jonathan Téllez Girón, Matías Alvarado

Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, CDMX, México

jtellez@computacion.cs.cinvestav.mx, matias@cs.cinvestav.mx

Resumen. En este artículo el lenguaje formal del futbol soccer es generado por una gramática libre de contexto, traducción de las reglas del juego, tal que las cadenas del lenguaje describen cualquier combinación posible de jugadas correctas, y así, un partido completo. La interacción entre los jugadores en el campo de juego conlleva un sistema de cómputo concurrente. Mediante centenares de simulaciones computacionales se valora el funcionamiento y desempeño del modelo y el sistema planteado. En particular, se aplica a las formaciones clásicas de un equipo al jugar bajo cierto esquema táctico y estratégico. Se analiza la aplicación del equilibrio de Nash para la selección de estrategias en el juego.

Palabras clave: Futbol, estrategias, equilibrio de Nash, sistema concurrente.

Football: Formal Language and Computable Simulation

Abstract. In this article, the formal language of soccer is generated by a context-free grammar, translation of the rules of the game, such that the language chains describe any possible combination of correct plays, henceforth a complete match. The interaction among the players on the playfield it comprises a concurrent computation system. Through hundreds of computational simulations the model pertinence and the system performance are evaluated. Particularly, it applies to the classic formations of a team playing under a certain tactical and strategic scheme. The applicability of Nash equilibrium to selection of strategies in the game is outlined.

Keywords: Football, strategies, Nash equilibrium, concurrent system.

1. Introducción

El futbol (del inglés *football*) es un juego estratégico de equipo, muy popular a nivel mundial. Se juega en una cancha de césped de 120 x 60 m, con arcos o porterías en cada lado del campo, entre dos equipos de 11 jugadores cada uno; un partido inicia con

el balón a media cancha y un jugador del equipo que ganó el bolado para dar el primer toque al balón lo pasa a un compañero. Los roles de juego, acorde a su posición en la cancha respecto a la portería propia son delanteros, medios, defensas, y el portero, siendo la misión del portero evitar anotaciones de gol en su portería, y sólo él tiene permitido el manejo del balón con manos y brazos. Sobre el césped los jugadores deben conducir el balón con los pies y pueden controlarlo, asimismo, con piernas, cuerpo y cabeza, pero no con brazos ni manos, salvo para los saques de banda y el portero respetando las reglas asignadas. A base de conducción individual y pases del balón entre compañeros y de esquivar a los rivales, deben buscar introducir el balón en la portería rival para anotar goles, y gana el partido el equipo que anota mayor número de goles.

Las estadísticas de este deporte junto con las técnicas de predicción de resultados son de gran interés para entrenadores, jugadores, empresarios y aficionados. En este artículo se presenta la gramática que genera el lenguaje formal de este juego, similarmente a como se hace en el béisbol (BB) [1] y el futbol americano (FA) [2]. Este modelo da pie a la simulación algorítmica del juego y al sistema concurrente para ejecutar la continuidad interactiva entre los jugadores: cada jugador es un proceso – hilo en el sistema computacional concurrente (SCC).

Se desarrollan las reglas gramaticales para cada jugador, que son leídas por el autómata finito no determinista correspondiente. La transición entre jugadas, así como la lógica de acuerdo a las condiciones del campo, se definen mediante las reglas de la gramática formal, que traducen las reglas del futbol. La interacción de jugadores es a través de la sección crítica del sistema, es decir, las variables y recursos en común entre los hilos. Las instancias particulares del sistema concurrente utiliza la ocurrencia promedio de jugadas, dadas a su vez en función del rol en la cancha: portero, defensa, medio o delantero. Las formaciones más usuales para un equipo son: 4-4-2, 5-3-2, 4-3-3, correspondientes al número de defensas, mediocampistas y delanteros. En algunos ejemplos se utilizan datos estadísticos de la Liga Española en el torneo 2015-2016, en particular de jugadores destacados en equipos con mejor posición en la tabla de la liga. El futbol presenta una dinámica presenta una dinámica diferente respecto al BB y al FA, dada una mayor continuidad de interacción no pausada. Por conveniencia, se limitan cada intervalo de tiempo en el sistema a un minuto de juego por jugada, t = 1,…, T, donde T = 90 es el límite regular de un partido oficial de minutos de juego. En cada intervalo cada jugador decide que jugada realiza.

En la Sección 2 se desarrolla la gramática y el lenguaje formal del futbol, en la Sección 3 el sistema de cómputo concurrente y en la Sección 4 se da un análisis inicial de cómo aplicar el equilibrio de Nash para la selección de estrategias. La sección 5 da cabida a la Discusión y se cierra el artículo con la Sección de Conclusiones.

2. Lenguaje Formal

El *alfabeto* $\sum$ es un conjunto finito, no vacío de *letras* o *símbolos.* Formalmente un *autómata finito* es la 5-tupla:

$$\langle Q, \Sigma, \delta, q_0, F \rangle,$$

donde:

- Q: Conjunto finito no vacío de estados.
- Σ: El alfabeto
- $\delta: \mathrm{Q} \times \Sigma \rightarrow \mathrm{Q}$ La función de transición que especifica a que estado pasa el autómata desde el estado actual al recibir un símbolo de entrada.
- $q_0 \in Q$ es el estado inicial del autómata
- $F \subset Q$ es el conjunto de estados finitos del autómata

La *gramática formal* es la 4-tupla:

$$G = \langle \Sigma_T, \Sigma_N, S, P \rangle,$$

donde:

- Σ_T es el alfabeto de símbolos terminales
- Σ_N es el alfabeto de símbolos no terminales
- $S \in \Sigma_N$ es el axioma, símbolo inicial con el que la inicia la regla inicial.
- P es el conjunto de reglas de producción de la forma $u ::= v$, donde $u = xAy, x, y, v \in (\Sigma_T \cup \Sigma_N)^*$ y $A \in \Sigma_N$.

En la gramática una producción es de la forma $A ::= v$, donde $v = (\Sigma_T \cup \Sigma_N)^+$. Puede contener la regla $S ::= \alpha$, donde S es el axioma, si la gramática no es recursiva en S. A los lenguajes generados por este tipo de gramática se les denomina *lenguajes independientes (libres) del contexto (l.i.c.).*

En la Tabla 1 se muestran las reglas de la gramática así como de cadenas de acciones o estrategias básicas del lenguaje del futbol, generadas a partir de la gramática independiente del contexto.

Tabla 1. Reglas de producción de gramática formal para un jugador en el futbol.

Regla	Descripción
$P \rightarrow go\ G$	Jugador anota gol
$P \rightarrow rea\ P$	Jugador acierta un regate
$P \rightarrow rep\ P$	Jugador recibe penalti con balón
$P \rightarrow far\ P$	Jugador recibe falta con balón
$P \rightarrow fap\ P$	Jugador realiza falta con el balón
$P \rightarrow fuj\ FJ$	Jugador saca el balón del campo de juego
$P \rightarrow d\ SP$	Jugador despeja el balón
$P \rightarrow pl\ SP$	Jugador realiza un pase largo
$P \rightarrow pc\ SP$	Jugador realiza un pase corto
$P \rightarrow ce\ SP$	Jugador realiza un centro
$P \rightarrow t\ SP$	Jugador realiza un tiro
$P \rightarrow as\ SP$	Jugador asiste a otro jugador
$P \rightarrow ref\ SP$	Jugador falla un regate
$SP \rightarrow far\ SP$	Jugador recibe falta sin balón
$SP \rightarrow blse\ SP$	Jugador bloquea el balón

Regla	Descripción
$SP \rightarrow db\ SP$	Jugador bloquea un disparo
$SP \rightarrow pec\ SP$	Jugador comete un penalti
$SP \rightarrow rep\ SP$	Jugador recibe penalti sin balón
$SP \rightarrow mac\ F$	Jugador comete mano
$SP \rightarrow r\ P$	Jugador recupera el balón
$SP \rightarrow i\ P$	Jugador intercepta el balón
$SP \rightarrow blce\ P$	Jugador bloquea el balón
$FJ \rightarrow per\ SP$	Jugador pierde el balón al sacarlo del campo
$F \rightarrow adv\ SP$	Jugador recibe advertencia
$F \rightarrow sdu\ TA$	Jugador recibe tarjeta amarilla
$F \rightarrow sdd\ STA$	Jugador recibe doble tarjeta amarilla
$F \rightarrow sf\ TR$	Jugador recibe tarjeta roja
$TA \rightarrow per\ SP$	Jugador con tarjeta amarilla pierde el balón
$STA \rightarrow sa\ TR$	Jugador acumula tarjetas recibiendo tarjeta roja
$TR \rightarrow exp\ X$	Jugador es expulsado del juego

Con base en la gramática formal definida y las reglas de producción es posible generar cadenas de juego. Sin embargo como se mencionó previamente, las cadenas de juego generadas por la gramática describen el juego de un solo jugador. El sistema concurrente se encarga de simular la interacción entre las cadenas de jugadas, a través de la sección crítica del sistema. La concurrencia se administra iterativamente por todos los hilos de jugador, verificando en cada tiempo de juego las jugadas realizadas por el resto de los jugadores, de tal manera que se determinan las jugadas posibles. La comprobación iterativa mencionada mantiene el realismo de las cadenas para todos los jugadores. Un ejemplo de las cadenas se puede ver en la Tabla 2. En la Tabla 3 se ejemplifica una cadena de juego para una anotación.

Tabla 2. Cadenas de jugadas generadas en el SCC utilizando estadísticas/probabilidades de jugadores reales.

Jugador	Tiempo(t)							
	1	**2**	**3**	**4**	**5**	**6**	**…**	**90**
Andrés Iniesta	*fap*	*adv*	*mvsb*	*mvsb*	*mvsb*	*blce*	*...*	*fap*
Lionel Messi	*pc*	*mvsb*	*fap*	*adv*	*mvsb*	*mvsb*	*...*	*pc*
Luka Modric	*mvsb*	*mvsb*	*mvsb*	*r*	*pc*	*mvsb*	*...*	*mvsb*
Cristiano Ronaldo	*mvsb*	*mvsb*	*mvsb*	*mvsb*	*mvsb*	*mvsb*	*...*	*mvsb*

Consideraciones y restricciones: cada equipo intenta anotar goles y evitar que el equipo rival los anote, y gana el que más anote. Un equipo de futbol define formación ofensiva y defensiva, y un estilo de juego, definido por el entrenador. Para triunfar, el diseño y uso de la combinación de estrategias individuales es cada vez más usual en el futbol competitivo. Una estrategia colectiva adecuada incrementa la probabilidad del triunfo. Con base en la estrategia elegida se define el curso de acciones del equipo en un partido. El modelado formal de este deporte multijugador da sustento al razonamiento estratégico del juego y la selección de estrategias. En [3] cada intervalo de tiempo $(t, t+1)$ es un minuto del partido, $t = 0, \dots, T = 90$; en cada t solo puede

anotarse un gol o realizar una expulsión, y no se consideran factores externos que afectan al resultado del partido. Estas restricciones las incluimos en nuestra propuesta.

Tabla 3. Cadenas de jugadas representando una anotación en el SCC utilizando estadísticas/probabilidades de jugadores reales.

Jugador	**Tiempo(*t*)**						
	…	**14**	**15**	**16**	**17**	**18**	**…**
Andrés Iniesta	...	*mvsb*	*mvsb*	*blse*	*mvsb*	*fap*	...
Lionel Messi	...	*blce*	*t*	*mvsb*	*mvsb*	*mvsb*	...
Luka Modric	...	*mvsb*	*fap*	*adv*	*mvsb*	*mvsb*	...
Cristiano Ronaldo	...	*mvsb*	*db*	*rea*	*t*	*go*	...

3. Concurrencia

Un proceso computacional, proceso a partir de ahora, es una actividad asíncrona susceptible de ser asignada a un procesador, y es el elemento básico para diseñar y ejecutar un programa. Los estados de un proceso son: (1) ejecución: utilizando el CPU; (2) listo: temporalmente detenido para permitir la ejecución de otro proceso; (3) bloqueado: imposibilitando de ejecutarse hasta que un evento externo lo habilite.

Dos procesos son concurrentes cuando existe un solapamiento en la ejecución de sus instrucciones. Un programa concurrente es un conjunto de procesos ejecutables de manera simultánea. A la porción de código que queremos se ejecute de forma indivisible se le llama *sección crítica.* Las secciones críticas del sistema se ejecutan en *exclusión mutua*, y solo uno de los procesos debe estar en la sección crítica en un instante dado. Si dos procesos distintos estuvieran accediendo, al mismo tiempo, a una variable compartida para actualizarla, puede dar lugar a un abrazo mortal entre dichos procesos, y la ejecución del programa entraría en un ciclo interminable (bucle): la ejecución de un proceso espera la salida de otro proceso, y este a su vez espera una salida del anterior, en una reiteración infinita. En programación concurrente es esencial cuidar de no caer en un abrazo mortal. Y este es uno de los retos a salvar en la programación del SCC del futbol.

En [4] se introduce el SCC. Una jugada de pase se ejemplifica en la Fig. 1.

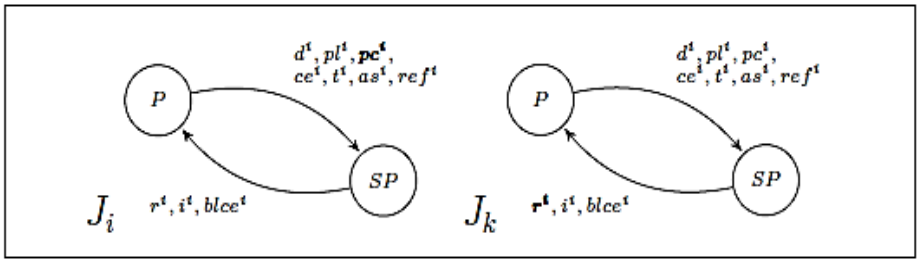


Fig. 1. Sistema concurrente de autómatas de dos jugadores para una jugada de pase corto [4].

4. Selección de estrategias

Para la elección de estrategias se asume la intención del entrenador y los jugadores de elegir las estrategias que dé lugar al mejor aporte de cada jugador en lo individual, pero, aún más, la que sea de mayor beneficio al equipo en lo colectivo, durante un juego. Así, la elección estratégica que proponemos utiliza el equilibrio de Nash (EN) [5]. Este método para elegir la estrategia tiene el efecto de minimizar el hacer jugadas poco exitosas o de riesgo para los jugadores en equipo durante un juego:

"Un perfil de estrategias es un equilibrio de Nash, si y solo si, para todo jugador y todas sus estrategias, la valoración del perfil actual es mayor o igual que el resto de los posibles perfiles"

Matemáticamente, es necesario definir una función de utilidad, por jugador, tal que evalué las estrategias del jugador con respecto al resto de los jugadores. Para formalizar el equilibrio de Nash introducimos la formulación de juego en forma normal.

4.1. Juego en Forma Normal

Sea $i \in \{1,2,\dots,N\}$ un elemento en el conjunto de jugadores; sea la estrategia pura σ_k^i la k-ésima acción del jugador i, tal que el conjunto de estrategias puras del jugador i:

$$D_i = \{\sigma_1^i, \sigma_2^i, \dots, \sigma_k^i\}.$$

Sea d un perfil de estrategia tal que:

$$d \in D, D = D_1 \times D_2 \times \dots \times D_N.$$

Sea la función de utilidad $U_i : D \rightarrow \mathbb{R}$ que asigna un valor a cada perfil de estrategia conforme el jugador i. Normalizando el valor $U_i : D \rightarrow [0,1]$.

Un juego en Forma Normal se define como:

$$G = \{D_1, \dots, D_N; U_1, \dots, U_N\}.$$

En un juego en forma normal se define una función de utilidad por jugador: cada jugador evalúa cada perfil de estrategias individualmente considerando su criterio. En el equilibrio de Nash, cada jugador, mediante su función de utilidad, evalúa los perfiles de estrategia y compara las evaluaciones entre sí.

4.2. Equilibrio de Nash

El equilibrio de Nash habilita un proceso de selección de perfiles de estrategia, siendo el criterio de selección de utilidad o máximo beneficio (menor riesgo) considerando las elecciones de los otros jugadores. Así, un perfil de estrategias $s^* = (s_1^*, \dots, s_N^*)$ satisface el EN sí y solo si:

$$\forall n, \forall s_n \neq s_n^* : U_n(s_n^*, s_{-n}^*) \geq U_n(s_n, s_{-n}^*).$$

La función de utilidad $U_i: D \rightarrow \mathbb{R}$ para el futbol, valora los perfiles de estrategia con respecto a tres factores principales:

- Probabilidad de ocurrencia promedio por minuto (OPM) de jugadas.
- Distancias relativas entre jugadores.
- Habilidad o eficiencia de jugadores.

La OPM es analizada en [6]: es un promedio de las jugadas efectivas entre el total, y puede personalizarse a cada jugador. La habilidad de los jugadores se pondera considerando las estadísticas del mismo. Las jugadas son valoradas y ordenadas basadas en frecuencias de ocurrencias estadísticas abstraídas de juegos reales de beisbol. A partir de las estadísticas se definen las probabilidades de ocurrencia de las jugadas. Con base en la probabilidad se calcula la ocurrencia factual, efectiva, de cada jugada.

Cada rol de juego considera habilidades características para él, y tal que habilite para un mejor desempeño durante el juego. Los dos factores principales a considerar para la función de utilidad son la valoración $V_i(s_k^i)$ del jugador i para la estrategia s_k^i, y la probabilidad $P_i(s_k^i)$ de su ocurrencia. La $P_i(s_k^i)$ de una estrategia se remite al estado actual del jugador. Esto es el proceso de elección aleatoria basado en la OPM. Cuando un jugador aplica su función de utilidad en un perfil de estrategias, está evaluado que tan benéfico es que el resto de los jugadores realicen la siguiente estrategia. Se propone una función de utilidad para cada rol de juego, similar a la definida en [2] para el futbol Americano. El valor obtenido a partir de un perfil de estrategias únicamente es comparable con los valores obtenidos por la misma función de utilidad.

5. Discusión y trabajos relacionados

En [7] el análisis estratégico del futbol es orientado a videojuegos utilizando algoritmos genéticos para generar oponentes que dependan de la habilidad del usuario y del nivel de juego. Un algoritmo genético utiliza técnicas inspiradas por la biología evolutiva tal como la selección natural, herencia, mutación y recombinación. La adecuación de cada estrategia del equipo es evaluada por la función de aptitud para enseñar como jugar de acuerdo a las reglas del futbol, y para evolucionar hacia mejores opciones estratégicas dadas por el valor de aptitud. En [3] la elección estratégica es iterando hacia atrás, desde el momento actual del partido t_i, $1 \leq i$.

En [1] se utiliza el equilibrio de Nash para identificar los perfiles de estrategia preferidos por todos los jugadores del equipo tal que aumente la probabilidad de éxito: la cooperación enfatiza una participación positiva de los jugadores con base en la estrategia definida para obtener el triunfo de un partido. Cada perfil de estrategias es evaluado por la función de utilidad de cada jugador, para determinar los perfiles de estrategia que potencien el éxito del equipo. Así, cada perfil de estrategia es analizado y se consideran los que cumplen el equilibrio de Nash. El equilibrio de Nash toma las

jugadas que sean más probables de ejecutarse con éxito, y así, mejora las opciones de éxito de un equipo.

Si bien el equilibrio de Nash toma las jugadas que sean más probables de ejecutarse con éxito, y así, mejora las opciones de éxito de un equipo. Computacionalmente, sin embargo, entre más jugadores participen en un juego y más estrategias puedan utilizar, la complejidad computacional de aplicar el equilibrio de Nash se incremente y pueda ser muy costoso. Más aun considerando la dinámica no pausada del futbol. Luego, aplicar el equilibrio de Nash, requiere combinarlo con heurísticas tal que disminuyan el dicho costo y sea factible utilizarlo para la toma de decisiones.

En [8], con el objetivo de reducir el tráfico en la red, [9] se enfoca en la aceleración del proceso de análisis por medio de la transferencia de equilibrio, o colocación de réplica (RPP). La transferencia de información sirve para determinar cuándo es conveniente replicar los datos dentro de un arreglo de servidores para acelerar el proceso que se atiende. A través de un análisis de costo de réplica y un proceso de minimización de costo de transferencia de objetos (OTC) se realiza la decisión estratégica para cada uno de los servidores. La función de utilidad determina el conjunto de estrategias factibles y se toma una decisión que minimice el costo de la réplica considerando a todos los agentes del sistema.

La idea clave en la transferencia de equilibrio es rehusar un equilibrio pre computado en juegos similares, ocurriendo después en el mismo estado si la pérdida de rehusar este estado puede ser tolerada. Para esto se deben tener en cuenta dos factores, la pérdida de la transferencia y la condición de transferencia. La pérdida de transferencia es la métrica de la desviación que puede surgir a partir de la reutilización de un equilibrio pre computado, la tolerabilidad de este factor debe ser configurada por el usuario, lo cual da a lugar a la condición de transferencia; si la perdida es menor al factor determinado por el usuario, entonces se puede realizar la transferencia.

En [9] el aprendizaje por refuerzo multiagente (MARL), es un proceso de decisión secuencial en problemas multiagente que en el entorno aprende a prueba y error. Los juegos de un solo tiro o no repetidos, como el futbol, en muchas ocasiones tienen un equilibrio similar en distintos momentos, lo cual permite reducir el número de cálculos por equilibrio, y así, la transferencia de equilibrio acelera el proceso.

Utilizando la transferencia de equilibrio, realizamos pruebas, simulando la posesión del balón y los pases entre jugadores. El campo de juego se representa en forma de una rejilla. Se limitan las acciones del juego a: arriba, abajo, izquierda, derecha y mantenerse quieto. De los experimentos realizados hasta ahora, pocas, pareciera eficiente aplicar la transferencia de equilibrio. Sin embargo, para sacar conclusiones se requieren muchas más pruebas.

6. Conclusiones

El lenguaje formal del futbol generado por una gramática libre de contexto habilita la descripción de cualquier combinación de jugadas y hasta de un partido completo. La interacción continúa es simulada por un sistema computacional concurrente. La habilidad futbolística de cada jugador y equipo se cuantifica en las estadísticas, las

cuales, a su vez, dan lugar a la definición de una distribución de probabilidades para jugadores y equipos. El equilibrio de Nash como herramienta clásica de toma de decisiones, inclusive cooperativas, es aplicable para seleccionar estrategias de juego en el futbol. El número de jugadores, de jugadas posibles para cada uno y la cuasi absoluta continuidad del juego, lo hacen computacionalmente costoso de simular de manera exhaustiva. Es necesario complementar esta selección de estrategias con heurísticas y métodos que simplifiquen esta complejidad.

Referencias

1. Alvarado, M., Yee Rendon, A., Cocho, G.: Simulation of baseball gaming by cooperation and non-cooperation strategies. Computación y Sistemas, Vol. 18, No. 4, pp. 693–708 (2014)
2. Yee, A., Rodriguez, R., Alvarado, M.: Analysis of Strategies in American Football Using Nash Equilibrium. In: International Conference on Artificial Intelligence, Methodology, Systems, and Applications (2014)
3. Dobson, S., Goddard, J.: Optimizing strategic behaviour in a dynamic setting in professional team sports. European Journal of Operational Research, Vol. 205, No. 3, pp. 661–669 (2010)
4. Téllez, J., Alvarado, M.: Concurrency Simulation in Soccer. Social Robotics, Vol. 9979, No. 0302-9743, pp. 961–970 (2016)
5. Nash, J.: Non-cooperative games. Annals of mathematics, pp. 286–195 (1951)
6. Téllez, J., Alvarado, M.: Modelado y análisis formal de jugadas del fútbol. Research in Computing Science, Vol. 113, pp. 147–156 (2016)
7. Fernandez, A. J., Cotta, C., Ceballos, R. C.: Generating emergent team strategies in football simulation. GAMEON, pp. 120–128 (2008)
8. Khan, U. S., Ahmad, I.: A Pure Nash Equilibrium-Based Game Theoretical Method for Data Replication across Multiple Servers. IEEE Transactions on Knowledge and Data Engineering, Vol. 21, No. 4, pp. 537–553 (2009)
9. Hu, Y., Gao, Y., An, B.: Accelerating Multiagent Reinforcement Learning by Equilibrium Transfer. IEEE Transactions on Cybernetics, Vol. 45, No. 7, pp. 1289–1302 (2015)

Impreso en los Talleres Gráficos
de la Dirección de Publicaciones
del Instituto Politécnico Nacional
Tresguerras 27, Centro Histórico, México, D.F.
noviembre de 2016
Printing 500 / Edición 500 ejemplares